Universidad Carlos III de Madrid

Trabajo Fin de Grado

Título: Desarrollo de un simulador de algoritmos de distribución de carga en sistemas distribuidos

Autor: Andrés Barber Buades

Titulación: Grado en Ingeniería Informática

Tutor: Félix García Carballeira

Fecha: 22 de Junio de 2016

Resumen

Desde hace años los sistemas distribuidos van cobrando mayor relevancia y están sustituyendo a los potentes mainframes. La aparición de estos sistemas se debe sobre todo a tres avances tecnológicos. El primero de ellos es el espectacular desarrollo de los microprocesadores, que ha permitido que cada vez sean más potentes y de menor coste. El segundo avance tiene que ver con el desarrollo de las redes de área local de alta velocidad, que proporcionan un medio de comunicación entre los ordenadores con una velocidad cada vez mayor. Y el tercer avance, pero no por ello menos importante, está relacionado con el desarrollo de software en los últimos años, que hace posible explotar estos sistemas.

Estos sistemas son utilizados por numerosas organizaciones que necesitan grandes capacidades de cómputo para sus proyectos, como por ejemplo, estudios de física, de química, de aviación, etc. El problema que vamos a abordar en este trabajo está relacionado con la distribución de la carga entre los diferentes nodos que componen un clúster, ya que la elección del algoritmo que se encarga de distribuir las tareas puede resultar en una mejora significativa del rendimiento global del sistema.

Por lo tanto, el objetivo de este proyecto es estudiar el comportamiento de diferentes algoritmos de distribución de la carga para una serie de clústeres definidos y determinar bajo qué circunstancias un algoritmo es mejor que otro.

Un clúster no es más que un conjunto de ordenadores unidos entre sí, a través de una red de alta velocidad, que se comportan como una única computadora.

Los algoritmos que se han tratado en este estudio han sido el *random*, *round robin*, *shortest queue first*, *two random choices*, y finalmente el *two random choices* con una serie de modificaciones, con el objetivo de observar si suponen una mejora del rendimiento. Todos estos algoritmos se han probado con los diferentes clústeres creados y con diferentes valores para la tasa de llegada, o lo que es lo mismo, las peticiones que llegan al sistema por segundo. Para implementar todo esto se ha utilizado SimGrid, una herramienta científica que permite el estudio del comportamiento de sistemas distribuidos tales como clústeres, redes de área local, sistemas P2P, etc.

Una vez finalizados los experimentos y recopilados los resultados, se han analizado y comparado entre ellos para poder así determinar cuál de los algoritmos propuestos presenta un mejor comportamiento ante diversas situaciones.

Abstract

For many years the utility and relevance of distributed systems has increased and they are progressively replacing the powerful mainframes. The rise of distributed systems is mainly due to three technological advances: Firstly, the spectacular development of microprocessors which has continued to bring progressive increases in power with a simultaneous reduction of prices; Secondly, the ongoing development of high speed local area networks, which provides a progressive improvement to the speed at which the computers communicate amongst themselves; And thirdly, but not least, is the development of specific software that has been happening over these past few years which makes the exploitation of these systems possible.

Distributed systems are utilized by numerous organizations for projects requiring very high and complex mathematical calculations, for studies in the fields of physics, chemistry and aviation for example. The problem to be addressed within this study is related to the distribution of load amongst the different nodes which compose a cluster; a cluster being nothing more than an assembly of computers linked amongst themselves by means of a high-speed network, which behaves like a unique computer. This is important as the selection of the algorithm responsible for distributing the tasks may result in a significant increase in the global performance of the system.

Therefore, the aim of this work is to study the behavior of different algorithms for distribution of the load for a series of defined clusters and to determine under which circumstances and conditions, one algorithm is better than another one.

The algorithms which are assessed in this study are *random*, *round robin*, *shortest queue first*, *two random choices* and finally the *two random choices* modified with the ultimate aim of observing an increase in their performance. All these algorithms have been tested on the different clusters created and with different values for the arrival rate or, in other words, the number of requests reaching the system every second. To implement all of this the SimGrid software tool was utilized which is a scientific tool that allows the study of the behavior of distributed systems such as clusters, local area networks and systems P2P.

Once the experiments were completed and the results gathered, they were analyzed and compared among themselves in order to be able to determine which of them presented the best behavior when facing diverse situations.

TABLA DE CONTENIDO

ÍNDICE DE GRÁFICAS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1: INTRODUCCIÓN

Los sistemas distribuidos han aparecido hace relativamente poco tiempo en la breve historia de la Informática. Los ordenadores cada vez son más pequeños y baratos, por lo que a medida que pasa el tiempo, caben más ordenadores en el mismo espacio a un coste más reducido. Hoy en día miles de ordenadores caben donde antes únicamente cabía uno. Su precio, al igual que su tamaño, también se ha reducido de manera exponencial y lo que es más importante, su rapidez y eficiencia han aumentado de manera vertiginosa. La comunicación por la red consume ciclos de reloj, por lo que un ordenador lento dedicaba la mayoría de su tiempo a esta tarea en vez de a ejecutar el programa de usuario. Por tanto, hace no muchos años, con el rendimiento y coste de las antiguas CPUs, la comunicación por la red no era viable. Sin embargo, gracias a los avances conseguidos en las redes de conexión, conectar ordenadores se ha convertido en algo muy fácil y barato.

Tanenbaum define un sistema distribuido como "una colección de ordenadores independientes que se presenta al usuario como un sistema único y consistente" (Andrew S. Tanenbaum, 2007, pág. 2). En esta definición se establecen dos puntos esenciales: el primero es el uso de la palabra "independientes", que significa que desde el punto de vista de sus arquitecturas, los ordenadores son capaces de operar de manera independiente; el segundo punto está relacionado con "un sistema único", lo que significa que el software permite a este conjunto de ordenadores interconectados mostrarse como uno solo ante los usuarios del sistema. Esto se conoce como el SSI (Single System Image), y como se verá más adelante en este estudio, su principal cometido es el de diseñar sistemas distribuidos fáciles de operar y mantener.

Ahora bien, ¿Por qué hacer uso de sistemas distribuidos? La utilización de sistemas distribuidos aporta las siguientes ventajas:

- Compartición de recursos; un sistema distribuido permite compartir recursos hardware y software.
- Apertura; suelen diseñarse sobre protocolos estándar que permiten combinar equipamiento y software de diferentes vendedores.
- Concurrencia; en un sistema distribuido varios procesos pueden operar al mismo tiempo sobre diferentes computadoras de la red.
- Escalabilidad; la capacidad del sistema puede incrementarse añadiendo nuevos recursos para cubrir nuevas demandas sobre el sistema.
- Tolerancia a fallos; la mayoría de sistemas distribuidos pueden proporcionar servicios incluso en caso de fallos de funcionamiento. Una completa pérdida del servicio ocurre únicamente cuando existe un fallo de funcionamiento en la red.

Pero los sistemas distribuidos no están exentos de problemas, y entre las principales desventajas del uso de una aproximación distribuida se pueden encontrar las siguientes:

- Complejidad; diseñar, implementar y utilizar software distribuido puede ser difícil. Los sistemas distribuidos son mucho más complejos que lo sistemas centralizados.
- La red puede perder mensajes y/o sobrecargarse; volver a cablear la red puede ser costoso y difícil.
- Seguridad; la facilidad de acceso desde cualquier punto crea problemas de seguridad. Debido a que el tráfico en la red puede estar sujeto a situaciones indeseadas, es más difícil asegurar la integridad de los datos.
- Manejabilidad; se requiere más esfuerzo para gestionar y mantener el correcto funcionamiento del sistema, ya que puede haber diferentes tipos o versiones de sistemas operativos; los defectos de una maquina pueden propagarse a otras máquinas con consecuencias inesperadas.

Dentro de los sistemas distribuidos, una de las arquitecturas más destacadas es la de los clústeres. El principal problema que presentan, que afecta considerablemente a su rendimiento, es la distribución de la carga, que se confía en gran medida a una serie de algoritmos. A través de éstos, los sistemas distribuidos obtienen el mayor grado de eficiencia posible. La distribución de la carga consiste en seleccionar el nodo o los nodos que ejecutan una determinada tarea, por lo que dependiendo de su elección, se consigue o no minimizar el tiempo de ejecución de las tareas. Esta es la función principal de este estudio, el análisis de diferentes algoritmos de distribución de la carga.

Para ello se ha definido un problema general (ilustración 1) que consiste en una entidad cliente que genera tareas y las envía a través de la red a la entidad *dispatcher*. El *dispatcher* recibe las tareas del cliente y dependiendo de lo que determine el algoritmo de distribución de la carga que implemente, enviará la tarea a uno u otro clúster.

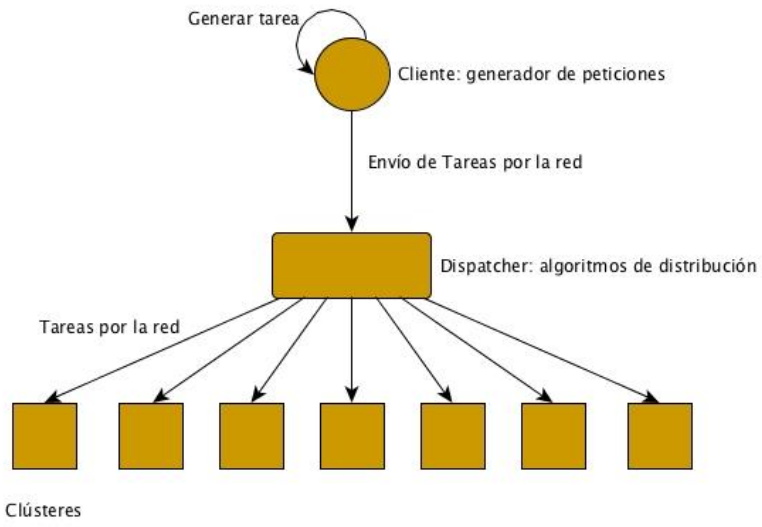


ILUSTRACIÓN 1: *PROBLEMA GENERAL*

1.1. OBJETIVOS DEL TRABAJO

Dado el problema de distribución de la carga en un clúster, el principal objetivo de este trabajo es estudiar y simular el comportamiento de diferentes algoritmos de distribución de la carga para analizar las diferencias entre ellos. Por tanto, en términos generales, los objetivos de una forma más concreta son los siguientes:

- **Estudiar los sistemas distribuidos, centrando la atención en los clústeres,** sus características y los componentes que lo forman.

- **Estudiar diferentes algoritmos que se pueden utilizar para distribuir la carga en sistemas distribuidos.** Para ello se estudiarán una serie de conceptos generales que distinguen unos algoritmos de otros.

- **Estudiar los conceptos básicos de simulación en sistemas distribuidos, como herramienta para guiar el diseño de los mismos.**

- **Realizar un estudio del *framework* de simulación SimGrid,** descubriendo sus características principales. Se desgranará su estructura en capas y se especificarán los elementos necesarios para realizar una simulación.

- **Definir diferentes ficheros de plataforma de clúster en SimGrid.** Se definirán varias configuraciones de clústeres diferentes y se implementarán en los ficheros de plataforma SimGrid.

- **Implementar los algoritmos estudiados en SimGrid y realizar la simulación con cada una de las plataformas definidas.** Los algoritmos escogidos se implementarán en SimGrid y se simularán con cada uno de los ficheros de plataforma creados. Cada simulación será ejecutada con diferentes valores para la tasa de llegada, o lo que es lo mismo, el número de tareas que llegan al sistema por segundo.

- **Obtener y comparar los resultados para determinar así qué algoritmos son los mejores y bajo qué circunstancias.** Con los datos obtenidos se crearan una serie de graficas que permitan estudiar la respuesta de los algoritmos respecto al tiempo medio de ejecución de una tarea y respecto al número de tareas medias esperando a ser ejecutadas.

1.2. ENTORNO SOCIO-ECONÓMICO

A medida que se avanza en el tiempo, cada vez son más los usuarios que acceden a servicios de internet a través de dispositivos móviles menos potentes, en lugar de hacerlo mediante los tradicionales PC de sobremesa. De continuar esta tendencia, como los usuarios no tendrán y tampoco estarán interesados en tener máquinas súper potentes, ¿Quién proveerá la potencia de cómputo? La respuesta a esta pregunta se encuentra en los sistemas distribuidos.

Antes se venían utilizando los Mainframes, computadoras centrales, potentes y costosas adquiridas en propiedad por las compañías, para cubrir sus necesidades de cómputo. En los últimos años se ha vivido una propensión hacia la sustitución de éstas supercomputadoras en favor de los sistemas distribuidos.

En los sistemas de información, la tendencia hacia los sistemas distribuidos nos lleva a derivar la capacidad de cómputo y el procesamiento de los datos hacia grandes centros de datos. Debido a la gran cantidad de información que éstos procesan, es muy importante encontrar los mecanismos necesarios para que realicen sus tareas de la manera más eficiente posible.

De este modo, invertir en el estudio de estos sistemas conduce finalmente a la mejora de su rendimiento, lo que supone a su vez una minoración de los recursos necesarios, al reducirse tanto el tiempo de ejecución por tarea como el coste de dicha ejecución.

Finalmente, destacar la importancia de crear simuladores que reproduzcan el comportamiento de estos sistemas. Como se ha comentado antes, el número de usuarios que hacen uso de los sistemas distribuidos no cesa de crecer año tras año, por lo que es necesario trabajar en la búsqueda de soluciones que permitan el aumento del número de usuarios al a vez que se preservan los beneficios obtenidos por su empleo.

1.3. MARCO REGULADOR

No existe ninguna regulación, normativa legal ni restricción que se aplique al contenido de este trabajo, ya que no se han utilizado datos de carácter privado y todas las herramientas empleadas para llevar a cabo el estudio son software libre.

El programa utilizado para realizar el estudio, como se verá con más detalle en el apartado 2.3 de este documento, ha sido SimGrid. SimGrid se distribuye bajo la licencia GNU GLP (General Public License). Los programas distribuidos bajo esta licencia no tienen ninguna restricción al uso. Es más, bajo los términos de esta licencia hay libertad de estudiar, compartir (copiar) y modificar el programa según sea necesario.

El Sistema Operativo sobre el que se ha instalado SimGrid ha sido Ubuntu en su versión 14.04. Este sistema operativo está basado en GNU/Linux y también se distribuye como software libre bajo licencia GLP.

Para realizar los diagramas de red se ha utilizado yEd. El programa yEd se distribuye bajo licencia *freeware*, que permite su uso ilimitado de forma gratuita, pero no autoriza su modificación ni su redistribución sin el permiso del fabricante.

La única licencia de uso que ha sido necesaria adquirir para realizar este Trabajo de Fin de Grado ha sido la de Microsoft Office. La adquisición de esta licencia está reflejada en el presupuesto del proyecto (apartado 5.2 de esta memoria).

1.4. ORGANIZACIÓN DEL DOCUMENTO

A lo largo de este documento se reflejará el trabajo y el estudio previo realizado para cumplir con los objetivos establecidos.

En este capítulo se han introducido los sistemas distribuidos, sus ventajas e inconvenientes, y el principal problema de distribución de la carga que presenta un sistema distribuido particular: los clústeres. A partir de éstos, se han establecido las bases y objetivos a cumplir en el estudio.

Para comenzar, se llevará a cabo un estudio del estado del arte, donde se analizarán los clústeres, prestando especial interés en sus características y en los componentes hardware y software que los componen. Dentro del estado del arte, se establecerá el problema que se quiere abordar con el estudio: la distribución de la carga en clústeres. También se estudiarán las características a tener en cuenta a la hora de elegir un algoritmo u otro para solucionar el problema de distribución de la carga, y finalmente, se describirá el funcionamiento de los algoritmos escogidos para solucionar dicha problemática.

El último aspecto que se tratará en el estado del arte tiene que ver con la herramienta que se utilizará para medir el rendimiento de los algoritmos seleccionados. Primero se introducirá a la simulación en sistemas distribuidos y a las herramientas disponibles que permiten realizar dichas simulaciones. Una vez seleccionada la herramienta que mejor se adapta a nuestras necesidades, se describirán y se establecerán los elementos básicos necesarios para realizar una simulación en la misma.

Una vez hecho esto, se explicarán en detalle las simulaciones que se han realizado y cómo ha sido su diseño e implementación en SimGrid, tanto de los algoritmos seleccionados como de las arquitecturas clúster definidas.

Después, se analizarán los resultados obtenidos con el objetivo de establecer la mejor o las mejores soluciones para una arquitectura dada bajo una serie de circunstancias, y se propondrán una serie de mejoras en base a las fortalezas o debilidades encontradas para cada uno de los algoritmos.

Finalmente, expondremos la planificación y el presupuesto que se han aplicado para realizar este proyecto, así como las referencias y la bibliografía consultada para documentarlo de la manera más precisa posible.

CAPÍTULO 2: ESTADO DEL ARTE

En este capítulo se va a hablar de los clústeres, del tipo de problema que se quiere abordar en este estudio, de los algoritmos existentes que podemos utilizar para solventar dicha problemática y de la herramienta que se ha utilizado para llevar a cabo el análisis.

El problema que se quiere abordar es un problema de distribución de la carga en sistemas distribuidos, concretamente en clústeres. Es decir, de cómo el rendimiento global del sistema se ve afectado dependiendo de la máquina a la que se le envía una tarea para su ejecución.

Una vez introducidos los clústeres, se nombrarán los diferentes algoritmos que se van a utilizar en el estudio, explicando las principales características de cada uno de ellos.

Seguidamente, hablaremos de las herramientas más destacadas para llevar a cabo una simulación de sistemas distribuidos. Una vez presentadas, hablaremos del *framework* SimGrid, utilizado para implementar las pruebas que realizaremos con los diferentes algoritmos en los diferentes entornos construidos.

2.1. CLÚSTERES

Básicamente hay 3 modos para mejorar el rendimiento:

1 Trabajar de manera más intensa, utilizando un hardware especial que permita reducir el tiempo por instrucción.
2 Trabajar más inteligentemente, optimizando algoritmos y utilizando técnicas de programación.
3 Servirse de la ayuda de múltiples computadoras para resolver el problema, ejecutando más instrucciones en el mismo tiempo.

Los clústeres explotarían principalmente la tercera idea ya que son un tipo de sistema de procesamiento paralelo o distribuido, compuesto por un conjunto de computadoras que trabajan de manera cooperativa como si se tratase de un único recurso de cómputo.

Estos sistemas presentan capacidades de red superiores a las de la red típica LAN y unos protocolos de comunicación con una latencia muy baja.

Los clústeres aparecieron como resultado de una serie de tendencias en Informática como son la redes de alta velocidad, disponibilidad de microprocesadores económicos de alto rendimiento, desarrollo de software de cómputo distribuido y necesidad de potencia computacional a gran escala para determinadas aplicaciones. Tienen una gran variedad de usos, desde aplicaciones de supercómputo y software para aplicaciones críticas, hasta comercio electrónico y bases de datos de alto rendimiento.

Están principalmente diseñados para ofrecer las siguientes características a un coste relativamente bajo:

- Alta disponibilidad: es la fracción de tiempo que el sistema está operativo en relación al tiempo total.
- Alto rendimiento: altas prestaciones en cuanto a capacidad de cálculo.
- Alta eficiencia: en Informática se refiere a la técnica usada para compartir el trabajo entre procesos, ordenadores, discos u otros recursos, lo que supone mayor cantidad de tareas ejecutadas por unidad de tiempo.
- Escalabilidad: capacidad del sistema para crecer sin aumentar su complejidad ni disminuir su rendimiento.

Por todo esto, los clústeres permiten a las diferentes organizaciones incrementar su capacidad de procesamiento manteniendo los costes bajos.

Por norma general, están compuestos de los siguientes componentes software y hardware:

- Nodos: pueden ser ordenadores, sistemas multiprocesador o estaciones de trabajo. Los nodos de un clúster pueden ser dedicados o no dedicados. Los dedicados son aquellos que no disponen de teclado, ratón ni monitor por lo que su uso está exclusivamente dedicado a realizar tareas que están relacionadas con el clúster. En los no dedicados, los nodos disponen de ratón, teclado y monitor. En este tipo de clúster se hace uso de los ciclos de reloj no utilizados por el usuario del computador, por lo que su uso no está exclusivamente dedicado a realizar tareas del clúster.

  Finalmente, cabe destacar la importancia de que los nodos de un clúster tengan características similares y que no se formen nodos heterogéneos debido a que podría afectar negativamente al rendimiento global del sistema.

- Software de base: sistema operativo multipropósito y multiusuario. Destinado a permitir una gestión eficaz y segura de sus recursos, gestionando el hardware de la máquina desde los niveles más básicos.

- Comunicaciones: los nodos de un clúster pueden conectarse mediante una simple red Ethernet o mediante tecnologías especiales de alta velocidad como Fast Ethernet, Gibabit Ethernet, SCI, ATM, Myrinet, etc.

- Middleware: reside entre el sistema operativo y las aplicaciones. Provee al clúster con el SSI (Single System Image), una interfaz única de acceso al sistema que permite que un clúster pueda ser visto como un único equipo. El Middleware también ofrece infraestructura para soportar SA (System Availability).

- Almacenamiento: puede consistir en una NAS, una SAN o almacenamiento interno del servidor. El protocolo más utilizado es NFS (Network File System), sistema de ficheros compartido entre servidor y los nodos, aunque también existen sistemas de ficheros específicos para clústeres como Lustre (CFS) y PVFS2.

- Programación paralela: permiten implementar algoritmos que hagan uso de recursos compartidos como CPU, memoria, datos, etc.

2.2. ALGORITMOS PARA DISTRIBUIR LA CARGA: CONCEPTOS BÁSICOS

Los algoritmos de distribución de la carga son los responsables de distribuir los trabajos según van llegando al sistema. En la actualidad existen varios algoritmos y el hecho de que los clústeres presentan diferentes problemas, entre los que se encuentran la heterogeneidad en las tareas y la diversidad de potencia entre nodos, provoca que no exista un algoritmo óptimo para todos los casos.

Debido a esto, hay una serie de conceptos generales que se debe de tener en cuenta y que distinguen unos algoritmos de otros.

- **Dinámico** o **estático**

    - Dinámico: Surgieron como contrapartida a los algoritmos de planificación estáticos que, a diferencia de éstos, realizan la planificación en tiempo de ejecución. Por ello, es recomendable su uso en problemas de paralelización, donde se desconoce de antemano el tiempo de ejecución de cada iteración o en los que el número de iteraciones no es fijo.

    - Estático: denominados también algoritmos preplanificados, en los que el cálculo del número de trabajos o iteraciones por nodo se realiza en tiempo de compilación.

- **Centralizado** o **distribuido**

    - Centralizado: en estas políticas la información se concentra en una única ubicación física que toma todas las decisiones de planificación. Esta solución tiene problemas de escalabilidad y suele presentar problemas de cuello de botella.

    - Distribuido: la información está repartida entre los distintos nodos y las decisiones son tomadas entre todos. Presentan problemas de coherencia de información y hacen un uso más intensivo de la infraestructura de red.

- **Política de transferencia**, determina cuando transferir.

    - Basadas en umbral: si la carga excede de T unidades de carga en el nodo S, éste se convierte en emisor de un proceso. Si la carga cae por debajo de T unidades, se convierte en receptor de procesos.

    - Tipos de transferencias:
        - Expulsivas: se pueden transferir los procesos ejecutados parcialmente (supone transferir el estado del proceso).
        - No expulsivas: los procesos en ejecución no pueden ser transferidos.

- **Políticas de selección,** selección del proceso a transferir. Se pueden seleccionar procesos nuevos o procesos que se encuentren en ejecución, poniendo especial atención en aquellos que cuenten con un tiempo de transferencia mínimo y en los que su tiempo de respuesta estimado en un nodo remoto es menor que el tiempo de respuesta local.

- **Política de ubicación,** selecciona el nodo al que transferir. Se muestrean otros nodos para encontrar uno adecuado. Los nodos pueden ser muestreados secuencialmente, en paralelo, de forma aleatoria, en función de su proximidad o de la información recogida anteriormente.

    - Broadcast: enviar un mensaje al resto de nodos.

    - Dos tipos de políticas: las iniciadas por el emisor y las iniciadas por el receptor.

- **Política de información,** información sobre otros nodos.

    - Bajo demanda: la información se recoge sólo cuando un nodo se convierte en emisor o receptor de procesos.

        - Iniciada por el emisor: el emisor busca receptores.
        - Iniciada por el receptor: el receptor solicita procesos a otros nodos.
        - Combinada: iniciada por el emisor o por el receptor.

    - Periódicas: los nodos intercambian información periódicamente, lo que tiene el inconveniente de sobrecargar el sistema, que no llega a adaptarse a las necesidades. Si el intercambio de información se realiza con alta frecuencia, se producirá mucha sobrecarga; por el contrario, una baja frecuencia supondrá un reparto de carga ineficaz. Todo esto hace difícil encontrar el punto intermedio óptimo.

- Dirigida por el cambio de estado: los nodos envían su información sólo cuando cambia su estado.
  - Centralizado: la información se envía a un nodo central.
  - Distribuido: la información es enviada a todos los nodos.
  - La recogida de información depende de la carga del sistema.

Todos estos conceptos y otros más son los que se deberán tener en cuenta a la hora de seleccionar el algoritmo de distribución de carga con el objetivo de obtener el mayor rendimiento global posible del sistema.

De entre todos los algoritmos existentes, hemos utilizado los siguientes para nuestro estudio: *shortest queue first*, *two random choices*, *random*, *round robin*, y *two random choices-round robin*. A continuación se van a describir las principales características de cada uno de ellos.

2.2.1. ALGORITMO *SHORTEST QUEUE FIRST*

El funcionamiento de este algoritmo es muy sencillo: cuando el nodo que se encarga de distribuir las tareas en el sistema recibe una tarea nueva, pregunta a todos los equipos del sistema sobre el tamaño de su cola, es decir, la cantidad de tareas que tiene cada nodo a la espera de ejecución. Los nodos contestan al nodo distribuidor, y éste envía la tarea al nodo con la cola de tareas más pequeña.

2.2.2. ALGORITMO *THE POWER OF TWO RANDOM CHOICES*

Para este caso, lo que se hace es seleccionar dos nodos aleatorios del sistema; el nodo que se encarga de distribuir los trabajos pregunta a esos dos nodos sobre el tamaño de su cola; finalmente, el nodo distribuidor selecciona el de la cola más pequeña como el destinatario de la tarea.

2.2.3. ALGORITMO *RANDOM*

En este algoritmo, cuando el nodo distribuidor recibe una tarea la envía a un nodo seleccionado de forma aleatoria. En este algoritmo no hay mensajes de información sobre tamaños de cola entre el nodo distribuidor y los demás nodos.

2.2.4. ALGORITMO *ROUND ROBIN*

El algoritmo *round robin* distribuye las tareas de forma equitativa y en un orden racional, comenzando con el primer host del primer clúster hasta llegar al último host del último clúster y empezando de nuevo desde el primero. Igual que en el algoritmo *random*, entre el nodo distribuidor y los demás nodos no hay intercambio de mensajes con información del tamaño de cola.

2.2.5. ALGORITMO *TWO RANDOM CHOICES-ROUND ROBIN*

Este algoritmo es una combinación del *two random choices* y el *round robin*. La filosofía es la misma que en el *two random choices*, pero en vez de coger los dos

servidores de forma aleatoria, se coge uno aleatoriamente y el otro por el algoritmo *round robin*.

2.3. Simulación de Sistemas Distribuidos

Una vez escogidos los algoritmos que se van a utilizar, se necesita de un entorno de trabajo, un *framework* que nos permita implementar las pruebas que se quieren realizar. Debido a las características técnicas del sistema a simular, se debe elegir un entorno que tenga la posibilidad de reproducir éstas, así como también el comportamiento del sistema real.

Dado que los sistemas son cada vez más complejos, el uso de *frameworks* es esencial a la hora de estudiar el comportamiento de los mismos. Gracias a éstos, los usuarios se pueden centrar en la creación de sus algoritmos, en analizar los resultados, en ejecutar simulaciones, incluso para entornos de gran tamaño y complejidad, sin tener que enfrentarse a la dificultad y muchas veces imposibilidad de ejecutarlas en entornos reales.

Por lo tanto, las máquinas virtuales se han convertido en una pieza fundamental para los entornos de computación distribuida. Es por esto que se necesita de un *framework* de simulación que permita el estudio de estos entornos a través de simulaciones exactas y que provea de APIs (Application Programming Interface) fáciles de utilizar.

Hay muchas herramientas disponibles para la simulación de sistemas distribuidos, siendo las más destacadas OMNet++, Parsec y SimGrid. La primera que se descartó para este estudio fue OMNet++ por ser la que presenta mayor tiempo de simulación; entre las restantes se eligió SimGrid debido a su mayor comunidad de usuarios y disponibilidad de documentación, por lo que facilitaría la solución de posibles problemas durante el estudio. Además de los factores mencionados, la definición de arquitectura de clúster en SimGrid nos pareció más sencilla que en las otras dos herramientas.

2.3.1. SimGrid

SimGrid es un juego de herramientas de simulación de código abierto que permite ejecutar miles de máquinas virtuales y controlarlas del mismo modo que se haría en la vida real. Con SimGrid se pueden definir tareas con diferentes tiempos de ejecución y realizar la simulación de recursos basados en máquinas estándar. Las principales características del simulador SimGrid son: ofrece un modelo de programación y un nivel de abstracción ajustado a las necesidades del desarrollador, permite modelar y evaluar de manera eficiente algoritmos de planificación para sistemas distribuidos y además hace posible una simulación más realista en comparación con otros simuladores.

SimGrid es software libre. Está distribuido bajo la licencia GPL que permite estudiarlo, compartirlo y modificarlo. La primera versión estable disponible para su descarga data de 1998 y es posible su uso en sistemas operativos Unix, Mac OS X y Microsoft Windows.

Las pruebas de este estudio se ejecutaron utilizando la versión 3.12, disponible desde el 13 de octubre de 2015.

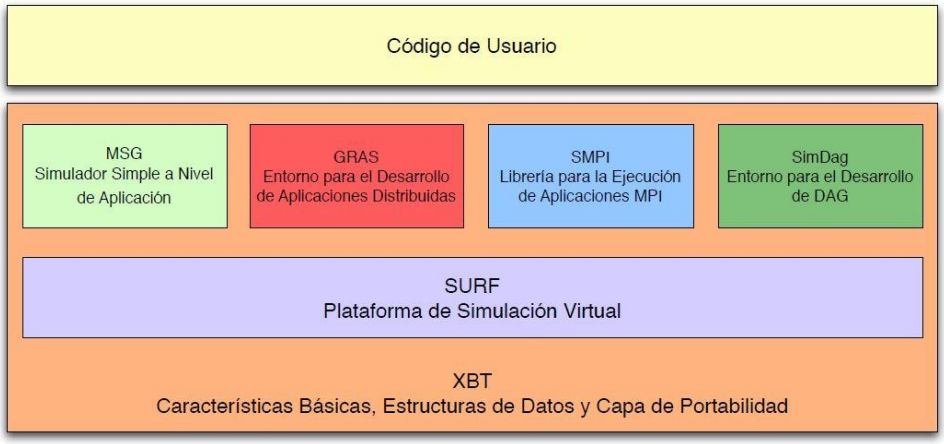


ILUSTRACIÓN 2: *ARQUITECTURA DE SIMGRID*

La arquitectura del simulador SimGrid sigue una estructura basada en capas. Básicamente se compone de tres capas principales: la capa de entornos de programación, la capa de simulación y la capa base.

- **Capa de Entornos de Programación**. SimGrid permite la simulación de paradigmas y entornos de programación construidos sobre un único núcleo de simulación. Además ofrece para cada tipo de paradigma de programación que se quiera desarrollar, un conjunto de componentes especializados. Entre estos componentes se pueden encontrar:

  - MSG. MSG es una interfaz introducida en la versión 2 que se utiliza para la simulación de procesos secuenciales concurrentes y con la cual los usuarios desarrollan sus algoritmos. Está compuesta por un conjunto de APIs para la simulación simple de aplicaciones. Provee de algunas entidades para modelar a los participantes en una simulación. A continuación se muestran los recursos ofrecidos en MSG:

    Process es el corazón del simulador. Está compuesto de código y datos que se ejecutan en un host.

    Mailbox es parecido al concepto de puerto en TCP. Es el lugar al que se envían y desde el que se reciben los mensajes. Se identifican con cadenas de texto y pueden tener funcionamiento síncrono o asíncrono.

Host es una máquina en la que se puede ejecutar un proceso. Por lo tanto, podrá ser representado como un recurso físico con capacidad de computación, algunos mailboxes para que los procesos en ejecución puedan comunicarse con otros procesos remotos y un conjunto de datos privados únicamente accesibles por los procesos locales.

Task es el trabajo a ejecutar o la cantidad de información a intercambiar. Puede ser ejecutada de manera local o transferida a otra máquina para su ejecución.

Link se refiere a los enlaces de red**.** Los hosts los utilizan para conectarse entre ellos. Por lo tanto, un link representa la conexión entre dos hosts, o entre un host y un switch. Un link se define por su latencia y su ancho de banda.

Utilizando estos recursos, un simulador se describe únicamente como procesos que se ejecutan en hosts e interactúan entre sí mediante el envío, el recibo o el procesamiento de tareas.

Los algoritmos ejecutados sobre SimGrid no deben de tener acceso directo a los links, y deben ser ejecutados como un proceso que envía una tarea a un host utilizando un mailbox. De hecho, un host puede tener muchos mailboxes, y un mailbox es identificado simplemente por una cadena de texto. Por lo tanto, enviar una tarea a un host usando un mailbox no es más que transferir la tarea a través de un link particular y ponerla en un mailbox determinado.

- GRAS. El componente GRAS ofrece una interfaz de programación completa que permite experimentar con aplicaciones distribuidas en sistemas heterogéneos, basados en *sockets* y llamadas a procedimientos remotos. Además de simular aplicaciones desarrolladas a través de su interfaz, también permite la ejecución de dichas aplicaciones en un entorno real, sin necesidad de modificarlas previamente.

- SMPI permite la simulación de aplicaciones escritas usando MPI, pero no ofrece una interfaz para el desarrollo de este tipo de aplicaciones.

- SimDag proviene de SimGrid v1 y se diseñó para la investigación de heurísticas de planificación, para grafos de aplicaciones paralelas. Es el homólogo al entorno de programación MSG, pero con el objetivo de simular tareas cuya planificación se realiza siguiendo el modelo DAG (Direct Aciclic Graphs).

- **Capa de Simulación.** También denominada SURF, constituye el núcleo de simulación de SimGrid y ofrece herramientas para simular entornos virtuales. Al estar escrito a muy bajo nivel no está pensado para uso directo por parte de los desarrolladores, sino como base para la simulación de los componentes que forman la capa de entornos de programación. Si el desarrollador determina que

algunos de estos componentes no se ajusta a sus necesidades, deberá utilizar SURF para implementar su propio simulador.

El núcleo de simulación de SimGrid implementa y provee de interfaces a un número de modelos de simulación que varían en sofisticación, y pueden ser usados para simular diferentes tipos de recursos (recursos de red, recursos de computación).

SURF provee de modelos para determinar el tiempo de la ejecución de acciones simuladas y el consumo de recursos de las mismas. Estos modelos pueden ser seleccionados y configurados en tiempo de ejecución, y cada uno es responsable de acciones y recursos como CPU, red, reloj... Por ejemplo, para los modelos de recursos de red, actualmente el modelo por defecto es el de redes TCP; también se tiene el modelo que descarga toda la simulación en el GTNetS, el modelo simple basado en la distribución aleatoria uniforme, y modelos más avanzados que utilizan la optimización de Lagrangian.

Seleccionando el modelo apropiado, el usuario puede intercambiar velocidad/escalabilidad por precisión, sin tener que cambiar el código de usuario.

Se puede acceder a todos los modelos de simulaciónca través de la función *surf_solve*.

- **Capa Base.** La capa base está constituida por el módulo XBT (eXtended Bundle of Tools), que consiste en el conjunto de herramientas necesarias para el desarrollo de componentes que se encuentran ubicados en las capas superiores. XBT es un *toolbox* escrito en ANSI C que implementa contendedores de datos, *logging* y excepciones y da soporte para configuración y portabilidad.

2.3.2. ELEMENTOS NECESARIOS PARA UNA SIMULACIÓN EN SIMGRID

Una vez que se ha presentado la arquitectura de SimGrid y con el objetivo de comprender de una manera más visual el funcionamiento del *framework*, se va a mostrar un ejemplo de los elementos necesarios para poder realizar una simulación.

En primer lugar, comentar que el código de usuario se puede escribir en C, Java o Lua. En el estudio se ha utilizado C, ya que durante la carrera principalmente se estudia Java, por lo que se pensó que ésta era una buena oportunidad para profundizar en los conocimientos de C.

Para ejecutar cualquier simulación en SimGrid se precisan 3 elementos: algo que ejecutar (el código de usuario), una descripción de la plataforma en la que se va a ejecutar la aplicación y finalmente, el pegamento que permite mapear los procesos simulados con la plataforma, o lo que es lo mismo, el fichero de despliegue.

- Código de usuario: aquí es donde se selecciona el API que se va a utilizar (SimDag, SMPI o MSG). Si la aplicación que se quiere crear es un DAG, se deberá utilizar SimDag. En caso de que se quiera estudiar código MPI, se utilizará SMPI. Para cualquier otro caso, se utilizará MSG. Como se ha comentado anteriormente, el código de usuario se puede escribir en C, Java o Lua.

- Plataforma: la definición de la plataforma se escribe en XML y aunque puede llegar a ser muy compleja, únicamente se van a mostrar los conceptos básicos en los que se basa.

  Hoy en día internet está compuesto por infinitas redes independientes. Estas redes son conocidas como Autonomous System (AS), subredes o simplemente redes LAN. El nombre de autónomo viene de que el enrutamiento (los caminos entre los diferentes nodos del sistema) se define dentro de los límites del AS. Si se quiere pasar de una red a otra, o lo que es lo mismo, de un AS a otro, se deberán utilizar los puntos de entrada (*gateways*). Dentro de un AS hay cables, routers, switches, ordenadores, etc.

  La descripción de una plataforma en SimGrid sigue la misma filosofía que la utilizada para el diseño de redes reales. Los ordenadores y otros elementos de red como cables, tarjetas, switches, etc, pertenecen a un AS. Dentro de un AS se puede establecer la ruta que se desee entre sus elementos. Para definir un AS se utiliza la etiqueta <AS>. Cabe destacar que dentro de un AS se pueden definir otros AS para establecer la jerarquía deseada.

  En cada AS se tienen disponibles los siguientes recursos: hosts (con su potencia de cómputo), routers, links (con sus anchos de banda y latencia) y clústeres, que contienen muchos hosts interconectados entre sí. Entre estos elementos, una ruta tiene que ser definida mediante el uso de una de las tres siguientes etiquetas:

  ASroute: utilizado para definir rutas entre dos AS.
  Route: para definir rutas entre dos hosts/routers.
  bypassRoute: para definir una ruta entre dos AS sin hacer uso del router que todo AS tiene creado por defecto.

  A continuación se va a mostrar una imagen de una posible arquitectura que podría definirse.

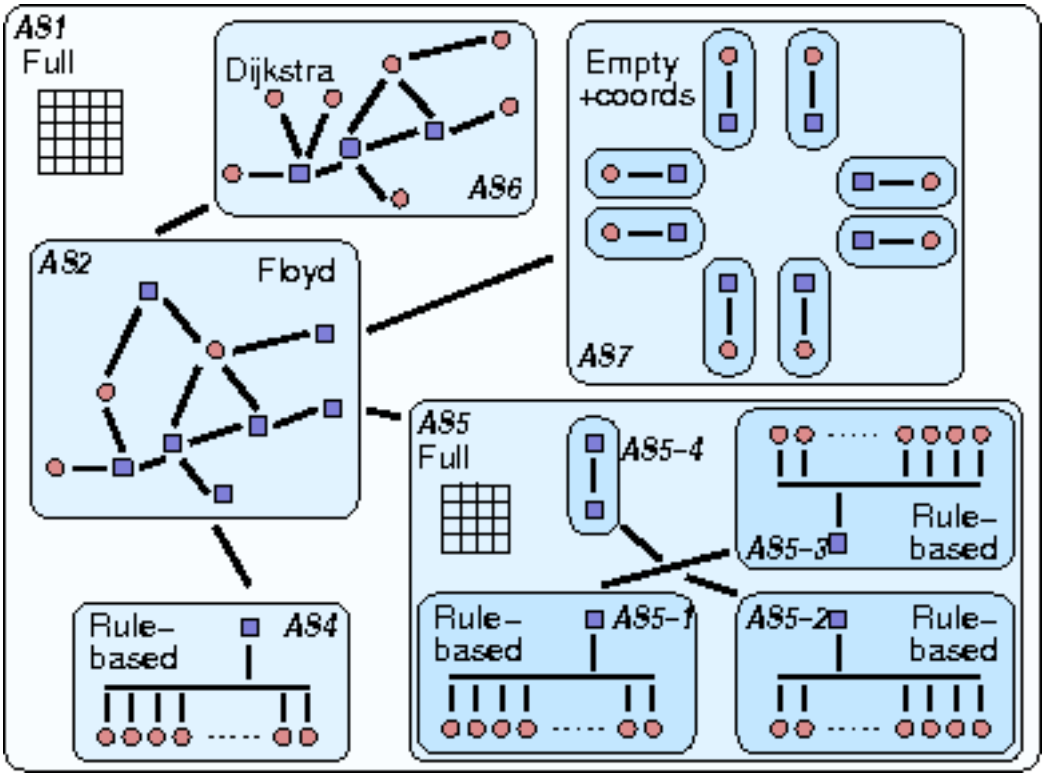


ILUSTRACIÓN 3: *EJEMPLO DE JERARQUÍA DE AS*

Los círculos rojos representan hosts mientras que los cuadrados azules, routers. Las líneas negras representan los links que permiten la comunicación entre AS o los elementos dentro del AS.

Estos son los conceptos básicos que hay que saber para definir el fichero de plataforma. Por lo tanto, una plataforma en SimGrid es una jerarquía de AS que contiene recursos.

```
<AS  id="AS0"  routing="Full">
   <host id="host1" power="1000000000"/>
   <host id="host2" power="1000000000"/>
   <link id="link1" bandwidth="125000000" latency="0.000100"/>
   <route src="host1" dst="host2"><link_ctn id="link1"/></route>
 </AS>
```

ILUSTRACIÓN 4: *EJEMPLO DE FICHERO DE PLATAFORMA*

La ilustración 4 es la descripción en el fichero de plataforma XML de un AS de nombre AS0 y que contiene dos hosts: host1 y host2. La conexión entre estos dos hosts se establece a través del link1, al cual se le ha establecido un ancho de banda de 1Gb/s y una latencia de 0,1 ms.

- Fichero de despliegue: consiste en establecer dónde ejecutan los procesos y qué argumentos de entrada debe tomar. La manera más fácil de comprender como

escribir un fichero de despliegue es a través de un ejemplo como el que se presenta a continuación.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <!-- The master process (with some arguments) -->
  <process host="Tremblay" function="master">
     <argument value="20"/>       <!-- Number of tasks -->
     <argument value="50000000"/>  <!-- Computation size of tasks -->
     <argument value="1000000"/>   <!-- Communication size of tasks -->
     <argument value="Jupiter"/>  <!-- First slave -->
     <argument value="Fafard"/>   <!-- Second slave -->
     <argument value="Ginette"/>  <!-- Third slave -->
     <argument value="Bourassa"/> <!-- Last slave -->
     <argument value="Tremblay"/> <!-- Me! I can work too! -->
  </process>
  <!-- The slave processes (with no argument) -->
  <process host="Tremblay" function="slave"/>
  <process host="Jupiter" function="slave"/>
  <process host="Fafard" function="slave"/>
  <process host="Ginette" function="slave"/>
  <process host="Bourassa" function="slave"/>
</platform>
```

ILUSTRACIÓN 5: *EJEMPLO DE FICHERO DE DESPLIEGUE*

El código XML de la ilustración 5 corresponde con un ejemplo del fichero de despliegue. Lo más importante de la etiqueta *process* es el atributo host, que establece en qué host será ejecutada la función; y el atributo *function*, que determina la función a ejecutar en el host seleccionado.

Para finalizar, cabe mencionar que en el estudio no se ha utilizado el fichero de despliegue porque SimGrid permite hacer el mapeo desde el código de usuario, en nuestro caso desde C.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

Una vez vistos los conocimientos básicos que incluyen los elementos necesarios para llevar a cabo una simulación en SimGrid, se va a describir analíticamente el diseño del sistema creado para este estudio.

En el punto 1 de este documento se ha definido un problema general con el que se pretende estudiar diferentes algoritmos de distribución de la carga. Partiendo del esquema general que se puede observar en la ilustración 6, se va a definir el diseño del sistema.

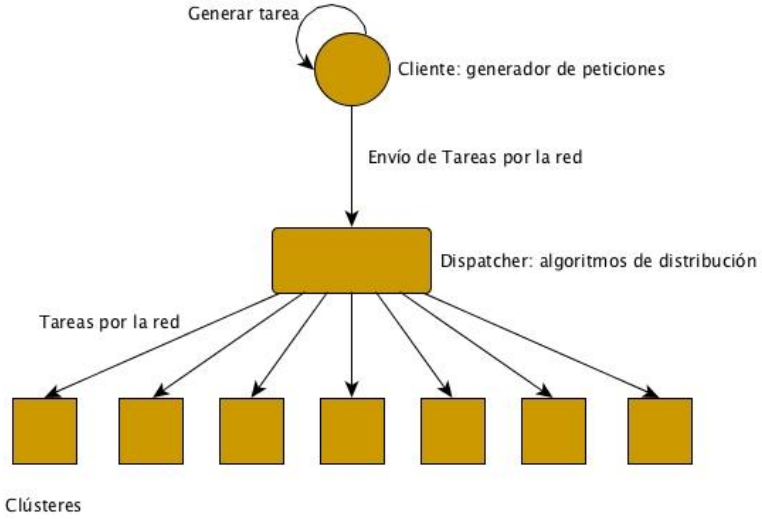


ILUSTRACIÓN 6: *PROBLEMA GENERAL*

Al objeto de comprender mejor lo que se quiere realizar, se deben de tener claros los diferentes elementos que se describen en la imagen:

- Generador de Peticiones: se ha llamado así al host desde el que se van a generar las tareas a ejecutar por el sistema. Las tareas generadas por este host son transmitidas al dispatcher.
- Dispatcher de Peticiones: host que recibe las peticiones generadas por el generador de peticiones y que se encarga de distribuirlas entre los diferentes hosts que componen el sistema. Aquí es donde se implementan los algoritmos que se van a evaluar en este estudio.
- Clúster: Son las máquinas que se encargan de ejecutar las tareas que reciben del dispatcher de peticiones.
- Redes de conexión: infraestructura por la que se envían las tareas.

Una vez descritos los diferentes elementos que componen el sistema se va a describir cómo se ha realizado su implementación en SimGrid. Como se ha descrito en el estado del arte, una de las interfaces que provee SimGrid es MSG, la cual está compuesta por un conjunto de APIs para la simulación de aplicaciones. Para el desarrollo del código de este estudio se ha hecho uso de MSG.

Como se ha descrito en el punto 2.3.2, para realizar una simulación en SimGrid se necesitan al menos tres elementos; el código de usuario, la plataforma del sistema y el fichero de despliegue. El código de usuario se ejecuta en los hosts, por lo que se deberán crear códigos específicos que ejecuten en cada una de las entidades descritas anteriormente las funciones que se esperan de ellas. La descripción de la arquitectura que cumpla con la funcionalidad establecida en la ilustración 6 se realizará en el modo establecido en el punto 2.3.2 para las plataformas del sistema.

Respecto al código de usuario se ha realizado la siguiente implementación para que cada host se comporte del modo deseado.

- Generador de peticiones: se encarga de generar las peticiones del sistema, para lo que se ha creado y asignado esta funcionalidad en el código. Por tanto, dentro del programa se tiene una función llamada "client" que el host generador de peticiones se encarga de ejecutar. La asignación de esa función al host se ha realizado mediante la llamada MSG_process_create_with_arguments(), en la que se especifica el host de la plataforma donde se ejecutará el código y el nombre de la función que ejecutará. Su ejecución se le asigna a un thread que se incluye en la lista de threads listos para la ejecución.

  El generador de peticiones ejecuta "client". En la práctica, la función es un bucle cuya finalidad es generar tantas tareas como se haya especificado. Dentro del bucle se acelera o retrasa la velocidad con la que se crean las tareas, dependiendo de la tasa de llegadas que se especifique a la hora de ejecutar la simulación (se explicará en el punto de "experimentación"). La tarea se crea mediante la llamada MSG_task_create(), momento en el que se le asigna un nombre, la cantidad de flops necesarios para su ejecución y un tamaño de tarea; además, se le asignará su hora de llegada al sistema como datos de usuario. Una vez creada la tarea, se envía de manera asíncrona al mailbox del dispatcher de peticiones a través de la llamada MSG_task_isend().

- Dispatcher de peticiones: El funcionamiento del dispatcher es sencillo, recibir tareas del generador de peticiones y distribuirlas entre los clústeres. Si se trata de una plataforma de un nivel, el dispatcher habla directamente con el host; si es de dos niveles, habla con otro dispatcher a nivel de clúster. La función que va a ejecutar el dispatcher toma ese mismo nombre e incluye el código de los algoritmos de distribución de la carga.

  La primera cosa relevante de esta función es MSG_mailbox_set_async() a través de la que se establece el mailbox del dispatcher asíncrono. Como se ha comentado con anterioridad, desde los mailbox se envían y reciben los mensajes. Una vez hecho esto, la función dispatcher es un bucle infinito cuya finalidad es recibir tareas, determinar su destino y enviarlas a su destino determinado. Dentro de este

bucle se reciben las tareas enviadas por el generador de tareas a través de MSG_task_recieve_with_timeout(). Cuando la tarea es recibida se crea una nueva con las mismas características, que es la que realmente se envía a los hosts. A esta nueva tarea se le asigna el tamaño que se generó en el generador de tareas para que el tiempo de generación de las tareas y su inserción en el sistema sea despreciable respecto al tiempo total de la simulación. Una vez hecho esto viene uno de los algoritmos que son objeto de este estudio:

- Random: la creación de random es muy sencilla ya que no implica tener que preguntar a los hosts sobre su carga. Por tanto, se genera un número aleatorio entre 0 y el número máximo de hosts en el sistema con uniform_int(). El número obtenido es el host al que se le enviará la tarea.
- Round Robin: al igual que random, en round robin tampoco hay intercambio de mensajes con los hosts sobre su estado. Para su implementación se ha definido una variable con valor 0 que corresponde con el primer host. La primera vez que el dispatcher de peticiones ejecuta el algoritmo la tarea es enviada al host 0 y esta variable es aumentada en una unidad; la vez siguiente que el algoritmo es ejecutado la tarea se enviará al siguiente host, y así hasta que la variable llegue al último host del sistema, momento en el que vuelve a ser 0 y se vuelve a comenzar.
- Shortest Queue First: para la implementación de este algoritmo ha sido necesario preguntar a los hosts sobre el estado de su cola. Este algoritmo consiste en un bucle que se encarga de preguntar a todos los hosts sobre su estado. Para eso, en cada iteración se crea una tarea de petición de tamaño con MSG_task_create(). La tarea es enviada al mailbox de todos los hosts a través de MSG_task_send(), que envía una tarea a un mailbox de manera bloqueante. Es decir, hasta que esa tarea no sea recibida en el host ejecutor, la ejecución no sigue su curso.

  Una vez hecho esto, hay que esperar la contestación de los hosts con el estado de su cola, para lo que se ha creado un nuevo un bucle en el que se reciben tantos mensajes como hosts haya. La recepción de los mensajes procedentes de los hosts se realiza mediante MSG_task_recieve_with_timeout(), mediante la cual se recibe de un mailbox una tarea de manera bloqueante; es decir, el flujo de ejecución será bloqueado hasta que se reciba una nueva tarea o se cumpla un determinado tiempo. En cada iteración del bucle se recibe la respuesta de un host; como los mensajes de respuesta de los host pueden no llegar en el mismo orden en el que fueron enviadas las tareas de petición de tamaño, las tareas de respuesta incluyen información del número de servidor y de su tamaño de cola. Para obtener la

información que transporta una tarea se obtiene a través de MSG_task_get_data(). La primera respuesta que se recibe es la que sirve de referencia para su comparación con las demás; si el valor de la cola del siguiente host que contesta es menor que el que se tiene como referencia, se sustituye por el que se acaba de recibir. Cuando todos los hosts hayan contestado, la tarea se enviará al host que ha contestado con un tamaño de cola menor.

- Two Random Choices: la implementación de este algoritmo podría describirse como una fusión entre el *random* y el *shortest queue first*. En primer lugar se seleccionan dos hosts aleatorios entre los que componen el sistema mediante uniform_int(). A continuación, se pregunta a los hosts seleccionados su tamaño de cola, del mismo modo que en el algoritmo *shortest queue first*, por medio de la creación y envío de una tarea de petición a los dos host determinados. Una vez realizadas estas preguntas se reciben los mensajes de contestación de la misma manera que la establecida en *shortest queue first*; una vez recibido los dos mensajes se determina el que tiene la cola más pequeña ya que será el receptor de la tarea.
- Two Random Choices 3: código exactamente igual al de *two random choices*, pero en lugar de hacer la comparación de la cola de dos host obtenidos de forma aleatoria, esta comparación se realiza con la cola de tres host.
- Two Random Choices 5: la comparación se realiza con las colas de cinco hosts seleccionados de forma aleatoria. Funcionamiento idéntico al *two random choices*.
- Two Choices RR: este algoritmo es una variación del *two random choices*. Se sigue comparando la cola de dos hosts, pero uno se obtiene de manera aleatoria y el otro por *round robin*. Para obtener el host aleatorio de entre de los del sistema, se hace uso de la función uniform_int(). Para obtener el otro host por *round robin* se define una variable a la cual se da el valor de 0. En la primera distribución de tareas el host por round robin es el 0, valor que aumentará en una unidad cada vez que el dispatcher distribuya una tarea. Cuando su valor sea igual al número máximo de hosts volverá a ser 0; de esta manera se obtiene un host de modo aleatorio y el otro por *round robin*. Una vez obtenidos los dos hosts a los que se preguntará sobre su cola, se crea la tarea tal y como se ha descrito en el algoritmo *shortest queue first*. Una vez creadas las dos tareas, se envían a los dos hosts seleccionados para solicitar el tamaño de su cola mediante MSG_task_send(). A continuación los dos hosts envían sus respuestas, que son recibidas a través de MSG_task_recieve_with_timeout(); una vez recibidas se determina a qué host se enviará la tarea dependiendo del tamaño de cola que presenten.

- Two Choices RR 3: se utiliza el mismo código que para *two choices rr*, pero en este caso se evalúa la cola de tres hosts, de los que dos son obtenidos de manera aleatoria y el tercero por *round robin*.
- Two Choices RR 5: la filosofía es la misma que la utilizada en el algoritmo *two choices rr*, sólo que en este caso se obtienen cuatro hosts de manera aleatoria y uno por *round robin*. Por tanto, se realiza la comparación del tamaño de las colas de cinco hosts.

Finalmente, cabe destacar que en los algoritmos en los que hay intercambio de mensajes con los hosts para conocer el estado de su cola, el dispatcher va a tener dos mailboxes, uno para recibir las tareas del generador de peticiones y otro para recibir las respuestas de los hosts con el tamaño de su cola.

Una vez que los algoritmos han determinado a qué host hay que enviar la tarea, su envío se realiza mediante MSG_task_send(), donde se especifica el mailbox del host receptor y la tarea que se envía.

- Clúster: cada host que compone un clúster ejecutará dos funciones, la función "server" y la función "dispatcherServer".

  La función "server" se encarga de recibir la tareas del dispatcher de peticiones. Lo primero que se realiza en esta función es establecer asíncrono el mailbox por el que se reciben las tareas. Una vez hecho esto, la función "server" es un bucle en el que a través de MSG_task_recieve_with_timeout() se van recibiendo las tareas. Cuando llega una tarea al host, se determina si se trata de una tarea de petición de tamaño de cola, o si por el contrario se trata del envío de una tarea para que sea ejecutada por el host.

  En caso de que se trate de una petición de tamaño, el host consulta el tamaño de su cola (número de tareas que tiene esperando y que está atendiendo), ejecuta el tiempo asignado a la tarea de petición a través de MSG_task_execute() y crea una nueva tarea de respuesta para enviársela al dispatcher de peticiones. En esta nueva tarea se incluirá el número de host y el tamaño de su cola. Una vez hecho esto, la tarea se envía al mailbox disponible en el dispatcher de peticiones que se encarga de recibir las respuestas de los host con su tamaño de cola. En caso de que la tarea recibida en la función "server" no se trate de una petición de tamaño, se entenderá que el host ha sido elegido para ejecutar esa tarea. Para este caso, lo que hace la función "server" es introducir la tarea en la cola de tareas pendientes de ejecución por el host. La inclusión de una tarea al final de esta cola se realiza a través de xbt_dynar_push(). Una vez hecho esto, el proceso que ejecuta "dispatcherServer" es activado y el bucle empieza de nuevo.

  La función "dispatcherServer" consiste en un bucle del cual se extraen las tareas encoladas por la función "server" para su posterior ejecución. Por tanto, en primer

lugar se coge el primer elemento de la cola de las tareas pendientes de ejecutar por ese host. La extracción de la cola se realiza mediante xbt_dynar_shift(). Una vez hecho esto, la tarea se ejecuta. Cuando la tarea se ha ejecutado completamente se calcula el tiempo invertido por el sistema en su ejecución.

Todo el código descrito anteriormente es aplicable para su uso en plataformas que tomen como base el problema general aportado por la ilustración 6 y que además dispongan únicamente de un dispatcher de peticiones. Como se verá en el punto 4 de esta memoria se propone una plataforma de dos niveles para la experimentación, en la que además de tener el dispatcher de peticiones como se ha referido anteriormente, se tiene uno dentro de cada clúster. Por tanto, el dispatcher que se ha descrito hasta ahora se encargará de distribuir las tareas entre los dispatchers de peticiones propios a cada clúster, siendo aquellos los que se encargan de repartir la carga dentro del clúster al que pertenecen.

Por tanto, dentro de los clústeres, en vez de tener todos los hosts disponibles para la ejecución, se tendrá uno que actuará de dispatcher de peticiones. Éste no ejecutará la función "dispatcher" descrita anteriormente, sino una versión adaptada de la misma. Además, algunas de las funciones descritas sufrirán variaciones para adaptarse a las necesidades de la plataforma de dos niveles.

A continuación se van a describir las funciones que han sufrido variación como también la nueva función que ejecutan los dispatchers de clúster:

- Dispatcher de peticiones: la primera función que se ha modificado para su uso en las plataformas de dos niveles ha sido la de "dispatcher". Su finalidad sigue siendo la de recibir tareas del generador de peticiones, pero ahora las tareas se distribuyen entre los dispatchers de peticiones de cada clúster. Por tanto, para las plataformas de dos niveles, este dispatcher ya no se comunica directamente con los host que componen un clúster, sino que lo hace con otro dispatcher, el dispatcher de peticiones propio de cada clúster. Las modificaciones que se han realizado a esta función para adecuarla a las plataformas de dos niveles han sido las siguientes:
    - El mailbox al que se le envía la tarea es el de un dispatcher de peticiones de clúster y no un host como se realizaba anteriormente.
    - Los algoritmos seleccionados para distribuir la carga entre los dispatcher a nivel de clúster han sido dos: *round robin* y *two random choices*. La implementación de cada uno de estos algoritmos se ha realizado de la misma manera que ya se ha descrito.

- Dispatcher de clúster: como se ha comentado, en la plataforma de dos niveles se va a incluir un host que va a realizar la función de dispatcher de peticiones en los clústeres. La función que ejecutan estos hosts no es la misma que la que ejecuta el dispatcher de peticiones descrito en los puntos anteriores, ya que su

funcionalidad no es exactamente la misma. A la función que ejecutan estos dispatchers de peticiones de clúster se le ha denominado "dispatcherCluster", y su finalidad principal es recibir tareas del dispatcher de peticiones y distribuir éstas dentro del clúster al que pertenecen. Es aquí donde se implementan los algoritmos que antes se aplicaban en el dispatcher de peticiones para las plataformas de un nivel.

Lo primero que se hace en esta función es establecer el mailbox por el que se reciben las tareas de manera asíncrona. Una vez hecho esto, el siguiente código es un bucle infinito en el que se reciben tareas procedentes del dispatcher de peticiones, de modo que cada vez que llega una tarea, se extraen los datos de usuario que contiene y se genera una nueva tarea copia de la anterior.

Puede darse el caso de que la tarea recibida sea una solicitud de tamaño de cola (en caso de que el algoritmo implementado en el dispatcher de peticiones sea el *two random choices*). En este caso, la función calculará el número total de tareas que están tanto en espera como siendo ejecutadas dentro del clúster y generará la tarea de respuesta correspondiente. En dicha tarea identificará el dispatcher de clúster que contesta y su estado; una vez hecho esto, contestará al dispatcher de peticiones para que éste tome una decisión respecto a qué clúster enviar la tarea finalmente. Si la tarea que se recibe no es una petición de cola, significa que ese clúster ha sido seleccionado para ejecutar la tarea, por lo que el dispatcher del clúster deberá asignar a uno de sus host la ejecución de la misma. Para determinar a qué host enviar la tarea, se hará uso de uno de los algoritmos que se han descrito anteriormente: *random*, *round robin*, *shortest queue first*, *two random choices*, *two random choices 3*, *two random choices 5*, *two random choices rr*, *two random choices rr 3* o *two random choices rr 5*. La implementación de estos algoritmos en el dispatcher de clúster ha sido prácticamente idéntica a la realizada en el dispatcher de peticiones para la plataforma de un nivel, con la única diferencia de que anteriormente estos algoritmos actuaban sobre todos los host del sistema y ahora, al encontrarse dentro del disaptcher de clúster, actúan únicamente sobre los host que residen dentro de un clúster.

- Clúster: los host del clúster siguen ejecutando las mismas funciones que las descritas para la plataforma de un nivel. La única función que ha sufrido variación para su uso en la plataforma de dos niveles ha sido la de "server". Este cambio está relacionado con las tareas de petición de tamaño de cola que llegan al host; en la plataforma de un nivel el solicitante era el dispatcher de peticiones y era a éste al que luego se le enviaba la correspondiente respuesta sobre el estado de la cola del host; ahora, el solicitante es el dispatcher de peticiones de clúster, por lo que es a éste al que se debe contestar. Por tanto, a la hora de crear el mensaje de respuesta, el mailbox de destino es el correspondiente al dispatcher de clúster. Para saber a qué dispatcher de clúster pertenece cada host, se compara el número

de host con el rango de hosts que gestiona cada dispatcher de clúster. Ahora que el host ya conoce a qué dispatcher de clúster pertenece, puede enviarle el mensaje de respuesta con el estado de su cola.

Para terminar con el código de usuario, se van a describir las funciones de MSG que se han utilizado para su creación.

Función	Descripción
MSG_process_sleep ()	Hace que el proceso se duerma el número de segundos que se le especifique.
MSG_task_create ()	Crea una nueva tarea. Se le especifica el nombre de la tarea, la cantidad de flops necesarios para ejecutar la tarea, la cantidad de bytes necesarios para transmitir la tarea y los datos contenidos por la tarea.
MSG_task_set_data ()	Permite incluir datos de usuario en las tareas.
MSG_task_isend ()	Envía una tarea a un mailbox determinado de forma no bloqueante.
MSG_mailbox_set_async ()	Hace que un mailbox determinado reciba las tareas de manera asíncrona.
MSG_task_recieve_with_timeout ()	Recibe una tarea de un mailbox. Esta llamada detiene el flujo de la aplicación hasta que se recibe una tarea o se cumple un determinado tiempo.
MSG_task_get_data ()	Devuelve los datos de usuario de una tarea determinada.
MSG_task_destroy ()	Acaba con una tarea. Únicamente el proceso al que pertenece esa tarea puede destruirla.
MSG_comm_destroy ()	Destruye una comunicación.
MSG_get_clock ()	Da el tiempo en segundos.
MSG_host_get_name ()	Obtiene el nombre de un host.
MSG_host_self ()	Indica la localización en donde el proceso está siendo ejecutado.
MSG_task_get_name ()	Permite obtener el nombre de una tarea.
MSG_task_get_compute_duration ()	Indica la cantidad de procesamiento necesario para la ejecución de una tarea.
MSG_task_send ()	Envía una tarea a un mailbox. Bloquea el flujo de ejecución hasta que la tarea es recibida en el otro extremo.
MSG_task_execute ()	Ejecuta una tarea y espera a su terminación.
MSG_init ()	Inicializa MSG.
MSG_main ()	Lanza la simulación MSG.

MSG_process_create_with_arguments ()	Crea y ejecuta un nuevo proceso. Se debe especificar un nombre, el código que debe de ejecutar, el host donde el proceso será ejecutado y dos argumentos de entrada.
MSG_task_set_compute_duration ()	Permite establecer la cantidad de recursos de computación necesarios para ejecutar una determinada tarea.
MSG_create_enviroment ()	Es el constructor de la plataforma. Se le pasa el xml donde se tiene descrita la plataforma.
MSG_function_register ()	Registra la función principal de un proceso en una tabla global.

TABLA 1: *FUNCIONES MSG UTILIZADAS*

Ahora que ya se ha descrito de manera detallada el código de usuario, se va a describir la implementación que se ha realizado del segundo elemento esencial en una simulación: la plataforma del sistema. La descripción de las diferentes plataformas del sistema se va a realizar de la forma establecida en el punto 2.3.2 de este documento.

- Plataforma de un nivel: esta plataforma se corresponde exactamente con la ilustración 6. Lo primero que se ha definido en la plataforma ha sido un AS dentro del que se va a describir toda la arquitectura deseada. En éste AS se ha incluido un router, que va a ser utilizado como punto de encuentro, ya que a través de él se interconectarán los diferentes elementos del sistema. Una vez hecho esto, se han definido dos clústeres compuestos únicamente por una máquina cada uno; uno de ellos va a hacer de generador y el otro de dispatcher de peticiones. Ahora que ya se tiene las entidad generadora y distribuidora definida falta crear las entidades ejecutoras, para lo se han definido clústeres, que en vez de estar compuestos únicamente por un host, lo están con un número determinado de hosts (se describirá en la experimentación el número de hosts que contendrá cada clúster como también las especificaciones técnicas de cada uno de ellos).

  Para poder enviar tareas de un lugar a otro es necesario definir un camino por el que circulen. Con esta finalidad los caminos se han definido con links, con tantos links como conexiones han sido precisas. Para las plataformas de un nivel descritas en la experimentación se emplea un link para conectar el generador de peticiones con el router que hace de punto de encuentro, otro para conectar el dispatcher de peticiones con el router punto de encuentro y otro por cada clúster que se añada para ejecutar tareas.
  Ahora que ya se tienen todas las entidades definidas, hay que interconectarlas a través del router, para lo que se ha utilizado ASroute, donde se especifican los nombres de las dos entidades que se quieren conectar entre ellas, el gateway de cada una de ellas y el link que se va a utilizar para la conexión. En SimGrid, cuando se utiliza "<cluster>" para generar un clúster, automáticamente se genera un router interno que actúa como gateway del clúster; todos los host que

pertenecen al clúster se encuentran conectados a ese router y la conexión se puede realizar a través de una red troncal (backbone) compartida por todos los host o mediante un cable independiente desde cada host al router. Para que un clúster use una u otra arquitectura, es preciso definir o no los atributos "bb_bw" y "bb_lat". En caso de que se especifiquen esos atributos en la etiqueta "<cluster>", el clúster se generará con un backbone con ancho de banda especificado por "bb_bw" y latencia "bb_lat". Como lo que se quiere en este estudio es que cada host se conecte directamente con el router de manera independiente, se ha optado por no especificar esos atributos.

- Plataforma de dos niveles: la diferencia entre la plataforma de un nivel y la de dos niveles es que esta última contiene un dispatcher de peticiones dentro de cada uno de los clústeres que se dedican a ejecutar tareas. Por tanto, la forma de describir los clústeres que van a ejecutar tareas en la plataforma de dos niveles se ha realizado de forma diferente, al haberse definido un AS por cada clúster dentro del que se ha incluido un host que va a ser el que haga de dispatcher de peticiones dentro del clúster; también dentro del AS se ha definido el propio clúster y un link, que se utilizará para conectar el clúster y el host que hace de dispatcher de clúster. Una vez hecho esto, para especificar que ese link es el que une el clúster con el host distribudior, se utilizará ASroute del mismo modo que se ha descrito en la plataforma de un nivel.

Para finalizar con las plataformas, se van a describir las diferentes marcas XML que se han utilizado para definir la arquitectura deseada.

Marca XML	Descripción
<AS>	Se ha utilizado para definir los sistemas autónomos. Se le debe asignar un identificador único y el modelo de enrutamiento que se desea. En este estudio se ha utilizado enrutamiento FULL, ya que permite definir las rutas de manera manual.
<router>	Crea un router con el identificador único que se le especifique. No ejecutan ningún código y no tienen ninguna influencia en el rendimiento global del sistema.
<cluster>	Permite definir de manera rápida un conjunto de hosts y su interconexión a través de una red interna. Se le debe especificar como mínimo un identificador, cómo nombrar a cada host que le pertenezca, la capacidad de cómputo de cada host, la latencia y el ancho de banda del cableado interno del clúster.

<host>	Crea un host. Se debe especificar como mínimo un identificador y la capacidad de cómputo en flops de cada core.
<link>	Crea un cable de red. Se identifica por su id, ancho de banda y latencia (esta última opcional).
<ASroute>	Permite establecer la rutas ente las entidades de manera manual.
<link_ctn>	Se utiliza para establecer el uso de un link en una ruta.

TABLA 2: *MARCAS XML UTILIZADAS*

CAPÍTULO 4: EXPERIMENTACIÓN

Una vez visto en el punto anterior el diseño del sistema utilizado, se van a describir las diferentes configuraciones creadas a partir del mismo y las evaluaciones de los algoritmos en esos sistemas.

Con el objetivo de determinar qué algoritmo de los descritos en el punto 2.2 se comporta mejor y bajo qué circunstancias lo hace, se han definido seis plataformas de un nivel y cuatro plataformas de dos niveles para la evaluación real y tres plataformas de un nivel y dos de dos niveles para la evaluación teórica. La diferencia entre las dos evaluaciones es que las propiedades de las plataformas que se utilizan en la real son reproducibles con la tecnología actual y en cambio, las de las plataformas que se utilizan en la evaluación teórica no, ya que tienen unas propiedades de conexión ideales que no pueden ser recreadas por la tecnología actual.

Por tanto, lo que se va a hacer en este punto, es describir las características de las plataformas creadas para lanzar las diferentes simulaciones, a través de las cuales se va a obtener el tiempo medio de ejecución de una tarea y el número de tareas medio esperando a ser ejecutadas en esa plataforma con un algoritmo de distribución de la carga determinado y una tasa de llegadas establecida. La tasa de llegadas se calcula mediante la multiplicación de 0.5, 0.7, 0.8, 0.9, 0.95, o 0.99 por el número de hosts que contiene la plataforma en la que se evalúa; cuanto mayor es el resultado de la multiplicación, menor es el tiempo entre la llegada de una tarea y la siguiente.

De las plataformas creadas para la evaluación real se han realizado dos modelos, uno con links de 1Gbps de ancho de banda y 100μs de latencia y otro modelo de 10Gbps y 50 μs de latencia. En cambio, para las plataformas utilizadas en la evaluación teórica, únicamente se ha descrito un modelo con links a 10^11GBps de ancho de banda y latencia 0μs.

Finalmente, recordar que el código que ejecuta cada una de las máquinas que componen las plataformas está descrito en el punto anterior.

4.1. PLATAFORMA PEQUEÑA DE UN NIVEL PARA LA EVALUACIÓN REAL

Dentro de este conjunto se ha definido únicamente una arquitectura, compuesta por 72 hosts divididos en tres clústeres. Por tanto, el primer clúster tiene los primeros 24 hosts del sistema, el segundo clúster los 24 siguientes y el tercer clúster, los últimos 24.

Como se puede observar en la ilustración 7, todas las conexiones a excepción de la del generador de peticiones se realizan a través de links de 1Gbps de ancho de banda y 100 μs de latencia o 10Gbps y 50 μs de latencia, dependiendo del modelo que se seleccione para la simulación.

Aunque en la imagen se ha representado al generador de peticiones de manera distinta, sigue siendo un clúster compuesto únicamente por un host. Como se puede observar en la imagen, el link que lo une con el router central no tiene las mismas características que los demás, ya que se quiere que el tiempo de generar las peticiones y su introducción en el sistema no afecte a la simulación.

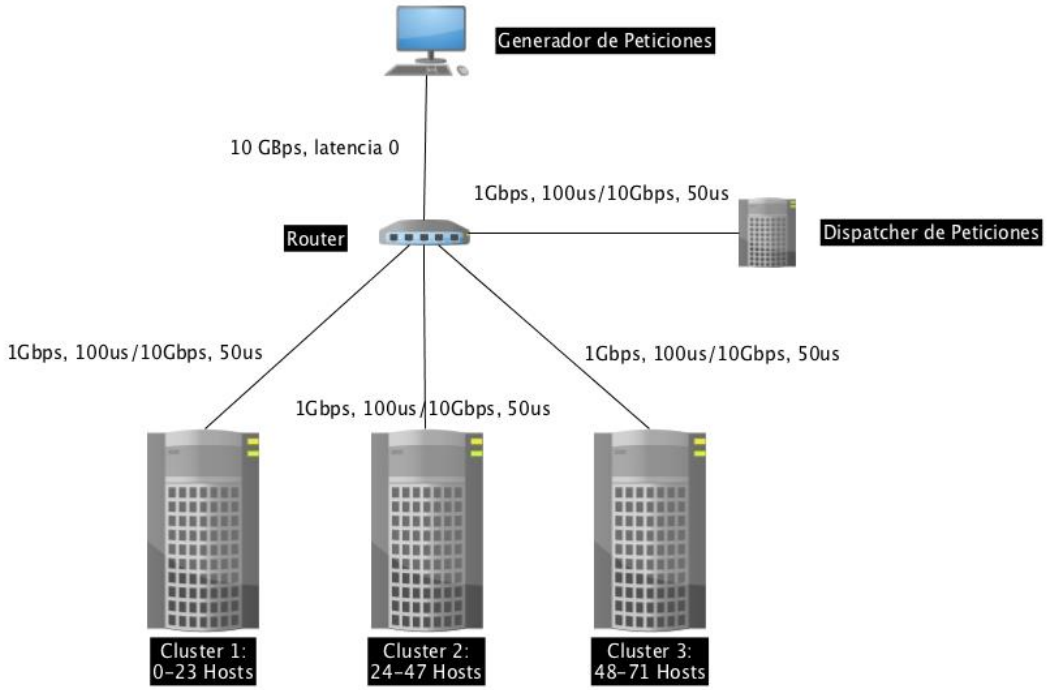


ILUSTRACIÓN 7: *PLATAFORMA PEQUEÑA*

El último aspecto a tener en cuenta en la definición de la plataforma está relacionado con la cantidad de Flops (Floating-point Operations Per Second) que son capaces de realizar las CPU (Central Processing Unit) de los hosts. Los hosts que componen los clúster 1, 2 y 3 son capaces de ejecutar 1GF por segundo, o lo que es lo mismo, 1.000.000.000 Flops por segundo, mientras que la máquina que contiene el dispatcher de peticiones es más potente y se le ha asignado una capacidad de cómputo de 2GF por segundo. Finalmente, la que contiene el generador de peticiones tiene 4GF por segundo.

4.2. PLATAFORMA PEQUEÑA DE UN NIVEL PARA LA EVALUACIÓN TEÓRICA

Esta plataforma es una copia de la descrita en el punto 4.1 con la única diferencia que todos los links pasan a tener un ancho de banda de 10^11GBps y una latencia de 0µs.

4.3. PLATAFORMAS MEDIANAS DE UNO Y DOS NIVELES PARA LA EVALUACIÓN REAL

Para este conjunto se han definido dos arquitecturas. La primera arquitectura es la que se define en la ilustración 8 y corresponde con una versión más grande de la arquitectura de un nivel definida en el punto 4.1; los clústeres pasan de tener 24 hosts a tener 256 y, en vez de tener 3 clústeres, ahora se tienen 10. Todas las demás características son exactamente las mismas.

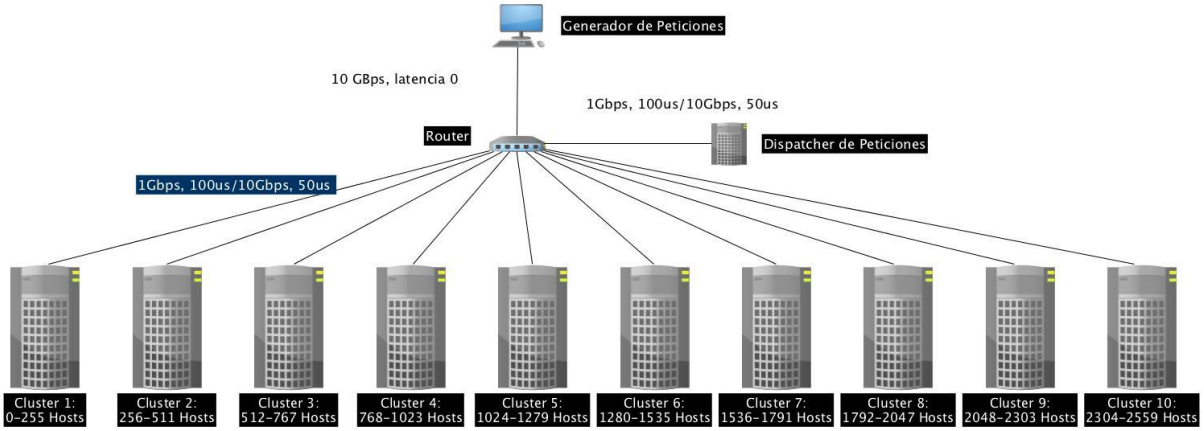


ILUSTRACIÓN 8: *PLATAFORMA MEDIANA DE UN NIVEL*

Para evaluar distribución de la carga en dos niveles se ha creado la arquitectura mostrada en la ilustración 9.

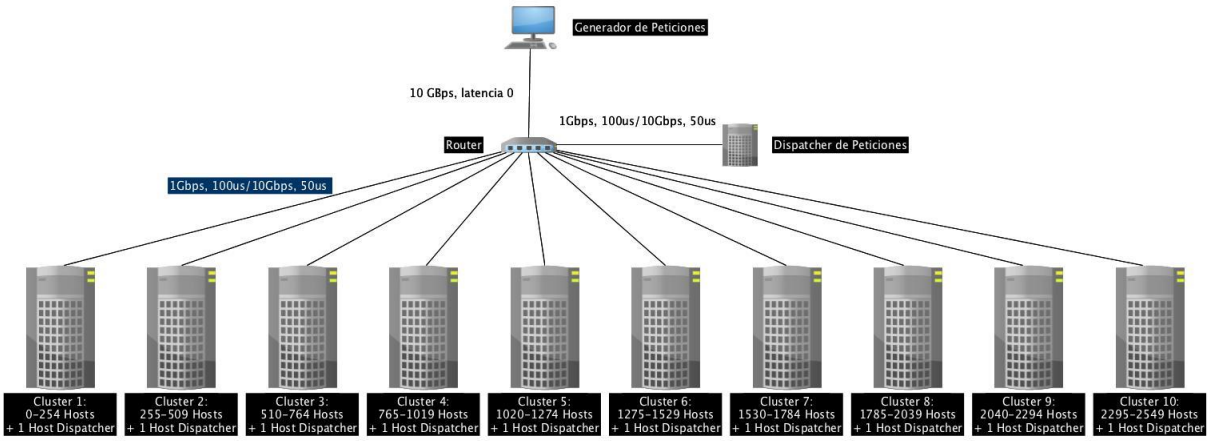


ILUSTRACIÓN 9: *PLATAFORMA MEDIANA DE DOS NIVELES*

Como se puede observar, la arquitectura es prácticamente igual a la descrita en la ilustración 8, pero en vez de tener 256 hosts disponibles por clúster, se tienen 255 ya que uno se comporta como de dispatcher dentro del clúster.

Al igual que en anteriores arquitecturas, las conexiones se realizan a través de links de 1Gbps de ancho de banda y 100 μs de latencia o 10Gbps y 50 μs de latencia, dependiendo del modelo que se seleccione para la simulación.

Las propiedades del dispatcher de peticiones, del generador de peticiones y de cada uno de los hosts que componen los clústeres son exactamente las mismas que las descritas en el punto 4.1.

4.4. PLATAFORMAS MEDIANAS DE UNO Y DOS NIVELES PARA LA EVALUACIÓN TEÓRICA

Se ha definido una plataforma para la evaluación teórica de la mediana de un nivel y otra para la evaluación teórica de la mediana de dos niveles. Las características técnicas de cada una de estas plataformas son exactamente las mismas que las descritas en el punto 4.3, con la diferencia de que ahora los links tienen un ancho de banda de 10^11GBps y una latencia de 0μs.

4.5. PLATAFORMAS GRANDES DE UNO Y DOS NIVELES PARA LA EVALUACIÓN REAL

Para este tamaño de plataforma también se han definido dos arquitecturas. La primera arquitectura se muestra en la ilustración 10 y corresponde con la versión grande de la arquitectura de un nivel definida en el punto 4.1; ahora se tienen 40 clústeres con 256 hosts cada uno, por lo que en total hay 10240 hosts para ejecutar tareas.

Las características de los equipos que forman el sistema se han mantenido invariables respecto a las definidas para la plataforma pequeña en el punto 4.1. Por último, recodar que dependiendo del modelo que se seleccione para la simulación, las conexiones se realizan a través de links a 1Gbps de ancho de banda y 100 μs de latencia o 10Gbps y 50 μs de latencia.

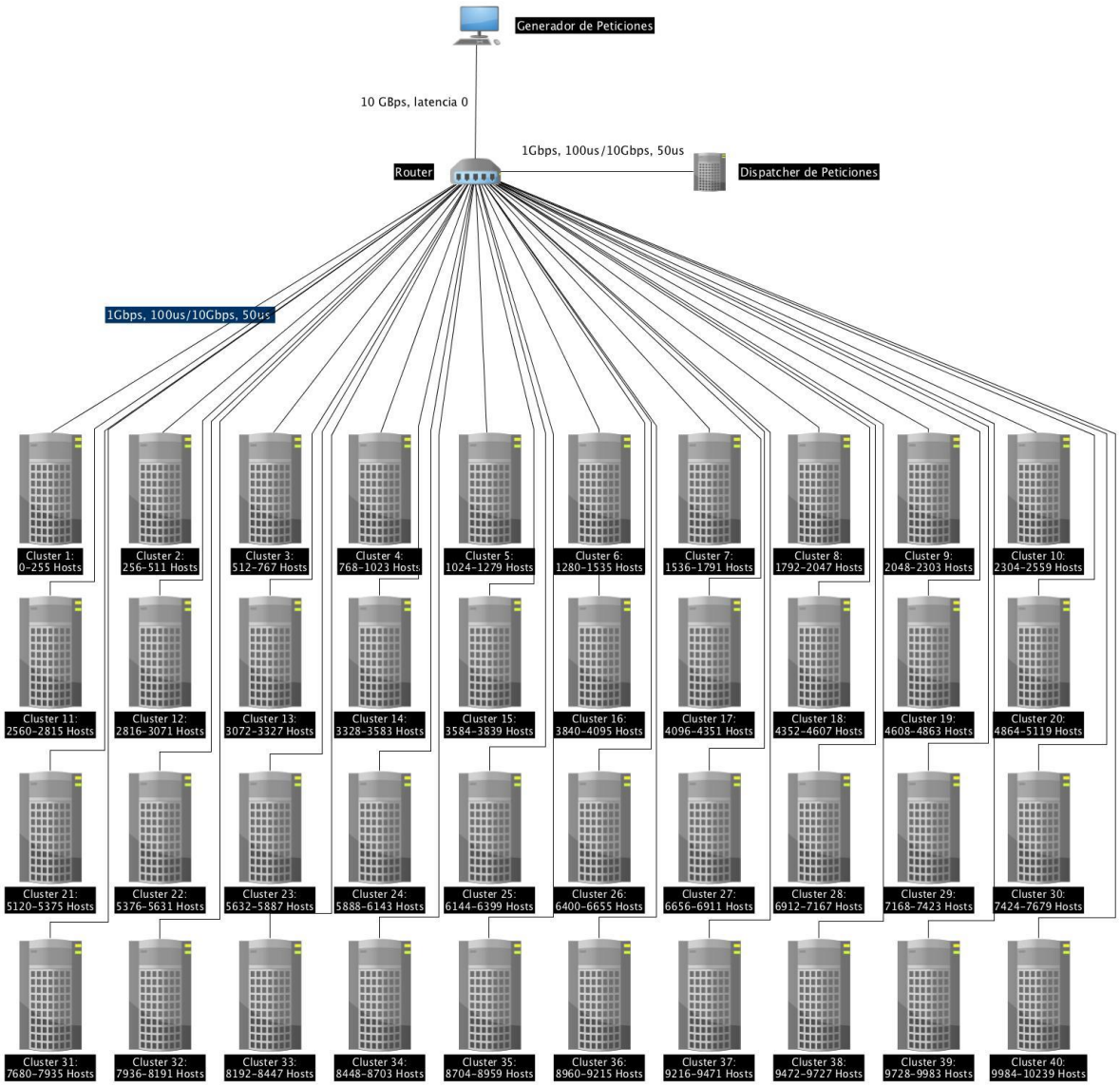


ILUSTRACIÓN 10: *PLATAFORMA GRANDE UN NIVEL*

La segunda arquitectura grande definida es una versión ampliada de la plataforma de dos niveles descrita en el punto 4.3; en vez de tener 10 clústeres ahora se tienen 40. Todo las demás propiedades, al igual que en las arquitecturas anteriores, permanecen invariables respecto a las descritas en el punto 4.1.

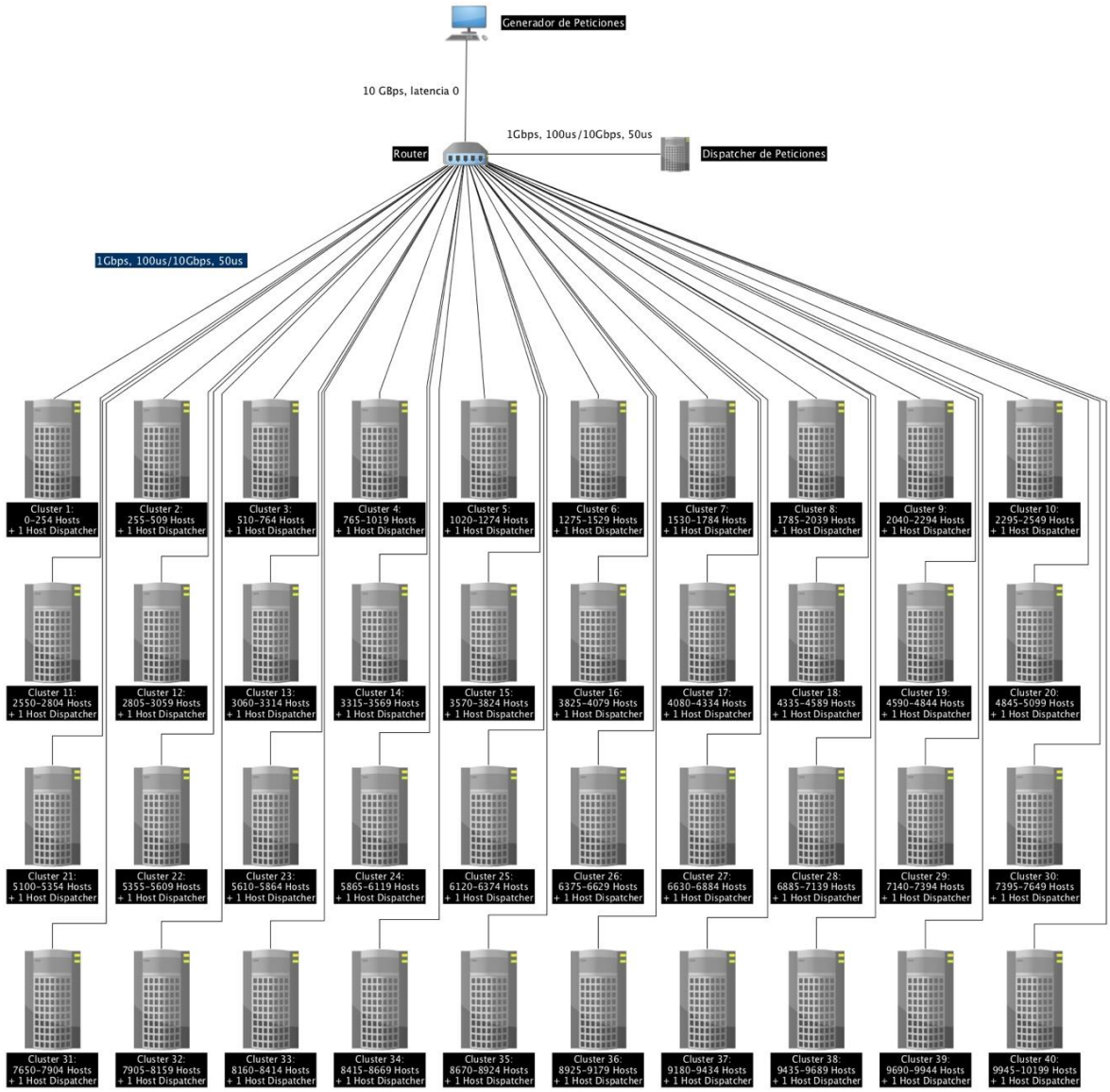


ILUSTRACIÓN 11: *PLATAFORMA GRANDE DE DOS NIVELES*

4.6. PLATAFORMAS GRANDES DE UNO Y DOS NIVELES PARA LA EVALUACIÓN TEÓRICA

Las características técnicas de las plataformas grandes de uno y dos niveles para la evaluación teórica son las mismas que las descritas en el punto anterior, aunque ahora las conexiones se realizan a través de links con un ancho de banda de 10^11GBps y una latencia de 0μs.

4.7. EXPERIMENTACIÓN CON LA PLATAFORMA PEQUEÑA DE UN NIVEL: EVALUACIÓN TEÓRICA Y REAL

Para la experimentación llevada a cabo con esta plataforma, el generador de peticiones crea 100.000 tareas a las que da a cada una un tamaño aleatorio comprendido entre 2048 Bytes y 51.200 Bytes. La cantidad de FLOPS necesarios

para ejecutar cada una de estas tareas se calcula de manera aleatoria y es la multiplicación de 1.000.000.000 por la variable pseudoaleatoria t, que se obtiene a partir de la fórmula siguiente: $t = -1 * \log(1-u)$, siendo "u" un número perteneciente al conjunto (0,1). Como se explica en el punto 3 de este documento, para evitar que la introducción de las tareas en el sistema influya en la simulación, se asigna un tamaño de 0 Bytes y 0 FLOPS de ejecución a las tareas enviadas desde el generador de peticiones al dispatcher de peticiones, siendo en este último donde se establecen las propiedades calculadas anteriormente.

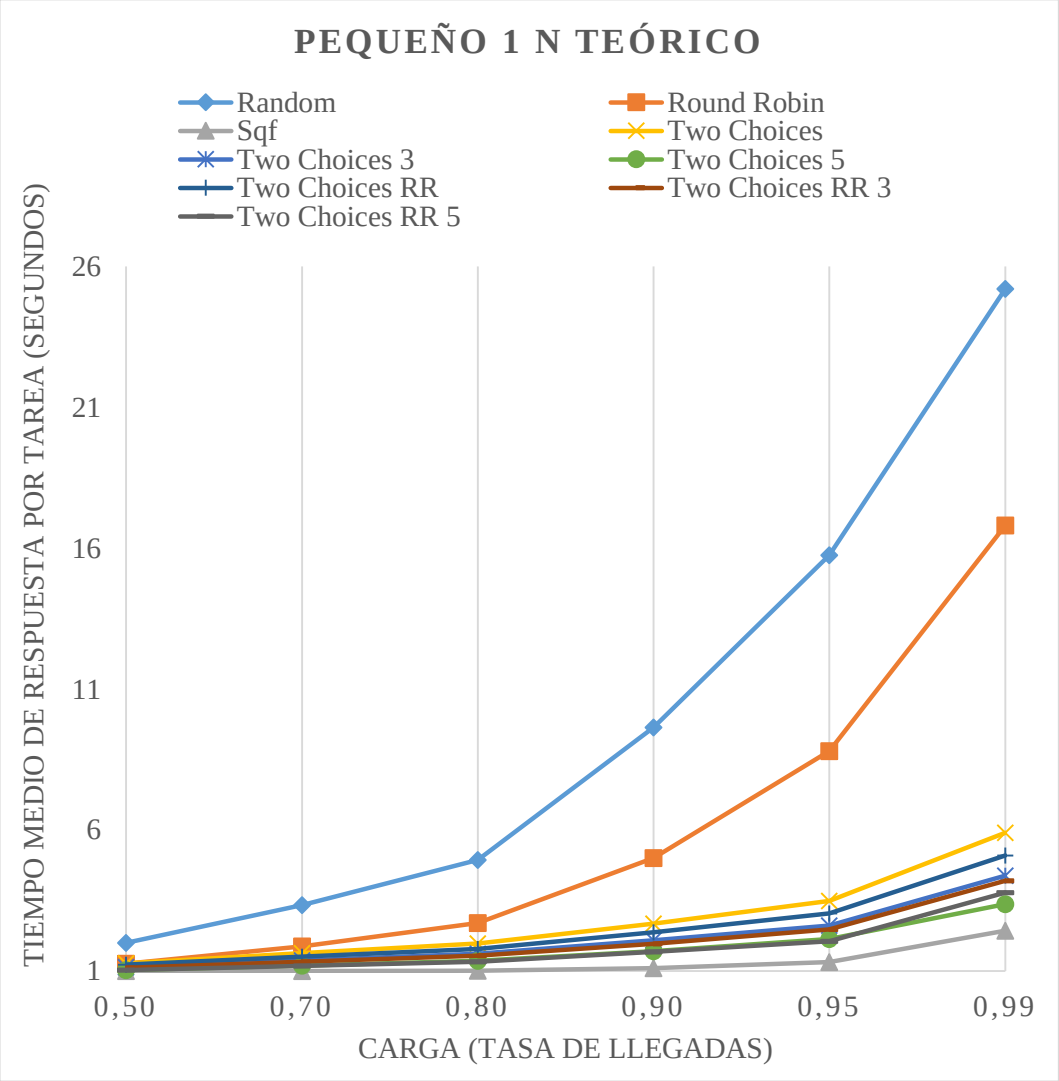
Como se describe en el punto 3, algunos de los algoritmos que se evalúan en este estudio realizan solicitudes sobre tamaños de cola. A estas tareas de solicitud, se les ha asignado 100 FLOPS de ejecución y un tamaño de 1024 Bytes, mientras que a las tareas que contestan a la petición, se les ha dado 0 FLOPS de ejecución y 1024 Bytes de tamaño.

Ahora que ya se ha descrito todo lo referente a la arquitectura y a las propiedades del sistema, se van a describir las diferentes evaluaciones realizadas en la plataforma pequeña. Es importante destacar que se han realizado 10 ejecuciones por algoritmo para cada tasa de llegadas.

Las gráficas 1 y 2 son el resultado de la evaluación teórica en la plataforma pequeña de los diferentes algoritmos objeto de este estudio. La primera representa el tiempo medio en segundos que lleva al sistema la ejecución de una tarea bajo una carga determinada, mientras que la segunda, la cantidad de tareas media esperando a ser ejecutadas en cada uno de los host de los clústeres.

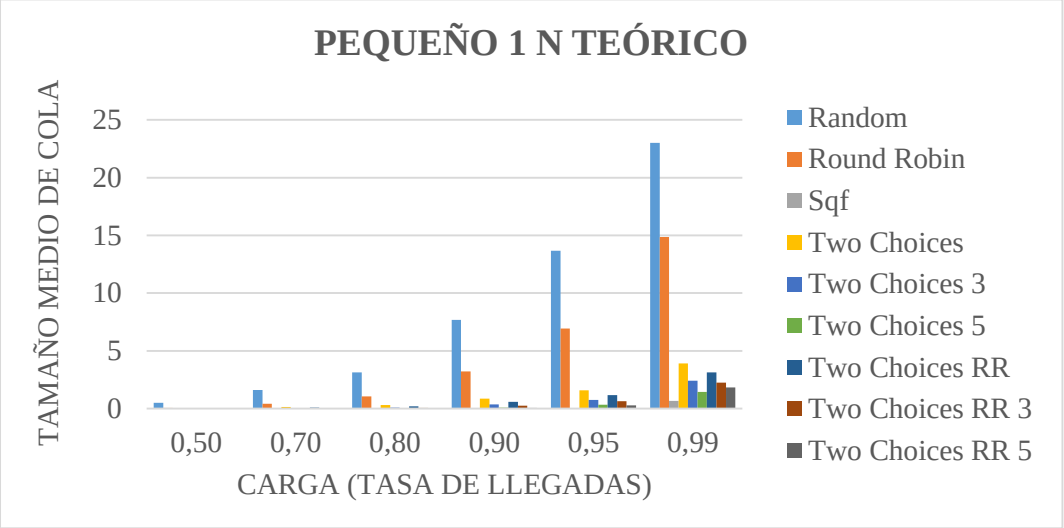
Como se puede observar en la gráfica 1, el algoritmo *random* es con el que se obtiene peor rendimiento y el *shortest queue first* con el que se obtiene mejor rendimiento. Esto se debe a que los mensajes de petición en la evaluación teórica tienen un coste despreciable, lo que permite al algoritmo *shortest queue first* seleccionar al host más desahogado sin apenas penalización.

El segundo aspecto que llama la atención de la gráfica es que el algoritmo *round robin* es el segundo con peor rendimiento. En un principio esto podría parecer contraproducente debido a que éste distribuye las tareas entre los hosts de manera incremental, pero justamente es eso lo que le penaliza, no tener en cuenta el estado de carga de los host a la hora de distribuir las tareas. Como se ha descrito anteriormente, la cantidad de FLOPS necesarios para ejecutar cada tarea es variable, por lo que se puede dar el caso de que la ejecución de una tarea en un host se demore mucho tiempo y mientras tanto *round robin* siga asignando tareas al host cuando podría habérselas enviado a uno menos congestionado y reducir así el tiempo de ejecución.



GRÁFICA 1: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA PEQUEÑA 1N*

Los algoritmos del tipo *two random choices* al igual que el *shortest queue first* también se basan en el estado de las colas para distribuir las tareas, pero a diferencia de éste último, la elección que realizan no es tan buena ya que comparan como máximo el estado de 5 host. Por tanto, debido a que las características técnicas de la red en la evaluación teórica permiten realizar peticiones sin apenas coste (únicamente el tiempo de ejecución de la tarea de petición), se puede afirmar que a medida que el algoritmo recaba información sobre el estado de las colas de más hosts mejor es el rendimiento del sistema. Otro aspecto importante a destacar es que, a medida que la tasa de llegadas aumenta, también lo hace el tiempo de respuesta medio de manera exponencial para el algoritmo *random* y *round robin* y de manera lineal para los demás algoritmos.



GRÁFICA 2: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA PEQUEÑA 1N*

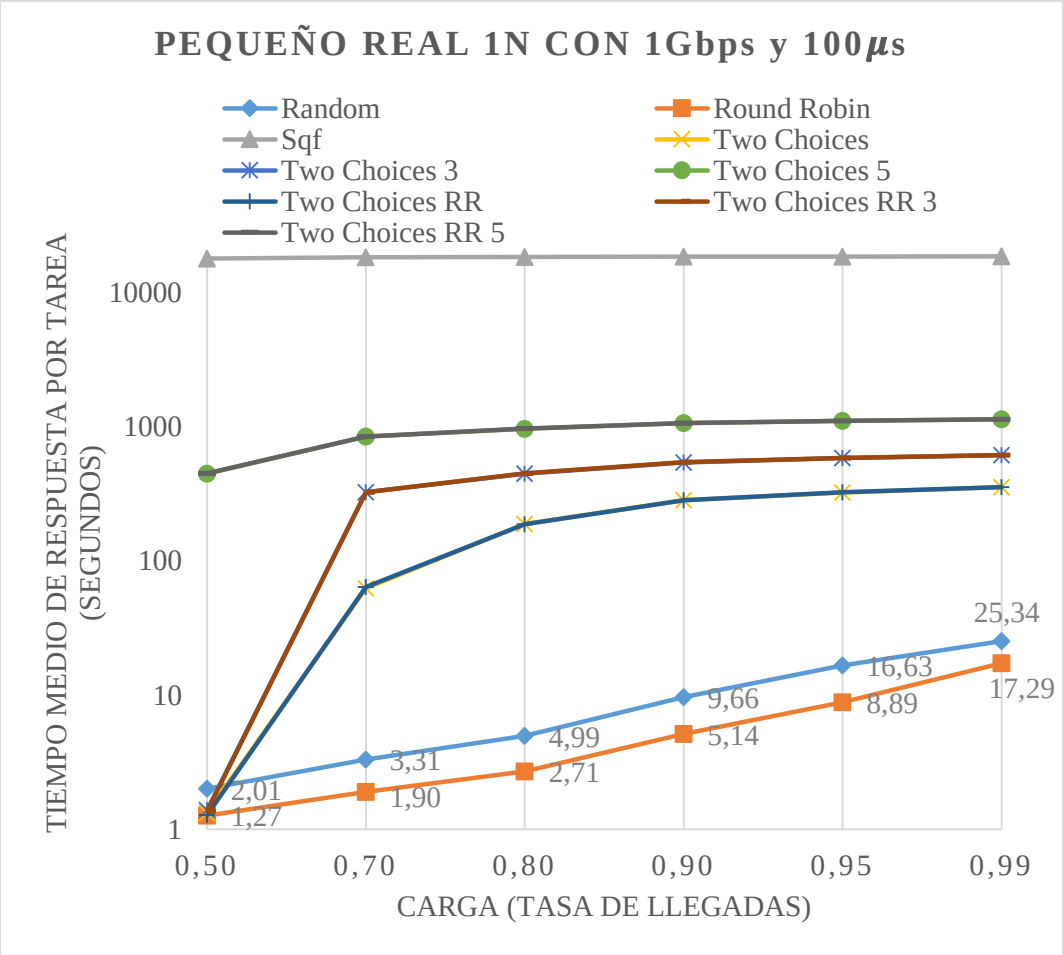
La gráfica 2 muestra el tamaño de cola medio en cada uno de los host de los 3 clústeres cuando se realiza la evaluación teórica en la plataforma pequeña. Como se puede observar, hay correspondencia entre los valores de tamaño medio de cola y el tiempo medio de ejecución por tarea aportado por la gráfica 1; aquellos algoritmos con los que se han obtenido los mejores tiempos medios de ejecución por tarea generan tamaños de cola pequeños en los host, mientras que aquellos con los peores tiempos crean colas más grandes, lo que se traduce en que los algoritmos que preguntan a los hosts sobre su cola distribuyen mejor las tareas, asignándolas a los hosts menos saturados, reduciendo así el tamaño de las colas y aumentando la cantidad de tareas ejecutadas por unidad de tiempo.

Además, cuando la tasa de llegadas es baja, el tiempo medio de respuesta por tarea es menor debido a que las tareas no consumen tiempo en las colas de lo host esperando a ser ejecutadas.

Una vez descrita la evaluación teórica, se van a examinar los datos obtenidos con la evaluación real en esta plataforma. La evaluación real de los algoritmos se ha realizado sobre los dos modelos descritos en el punto 4.1; el modelo cuyos links tienen un ancho de banda de 1Gbps y 100µs de latencia y el modelo con links de 10Gbps y 50µs.

Las gráficas 3 y 4 representan los datos obtenidos en la evaluación real con la plataforma pequeña para el modelo que utiliza links de 1Gbps. En la primera gráfica, a diferencia de lo obtenido en la evaluación teórica, el algoritmo *shortest queue first* es el que peor se comporta, mientras que el *round robin* es con el que se obtienen mejores resultados. Esto se debe a que transferir las peticiones y respuestas con el tamaño de las colas a través de la red ahora tiene un coste, por lo que ya no es eficiente preguntar a todos los host cada vez que una tarea llega al sistema. Además, el hecho de tener únicamente un dispatcher de peticiones lo empeora, ya que en la evaluación

real pasa la mayor parte del tiempo enviando y recibiendo mensajes con el estado de las colas y no distribuyendo las tareas por lo que hace de cuello de botella.

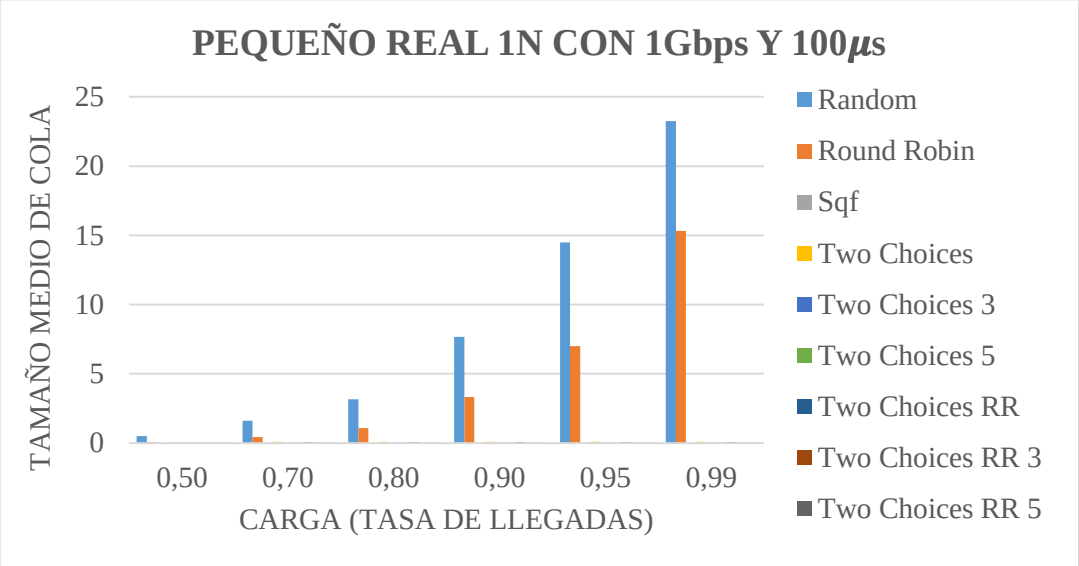


GRÁFICA 3: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA PEQUEÑA*

Otro aspecto a destacar es que no hay diferencia entre el rendimiento obtenido por el algoritmo *two random choices* y el obtenido por *two random choices–round robin.* Además, dentro de estos dos últimos, a medida que el algoritmo realiza más solicitudes sobre las colas, mayor es el tiempo medio de ejecución de una tarea en el sistema.

Para terminar de comentar la gráfica 3, destacar que debido a las características del sistema evaluado, no es adecuado el uso de algoritmos cuya funcionalidad se basa en información obtenida de los host mediante el intercambio de mensajes.

El tamaño medio de cola de la evaluación anterior se muestra en la gráfica 4 y como se observa, el tamaño de las colas ha variado respecto a la evaluación teórica.



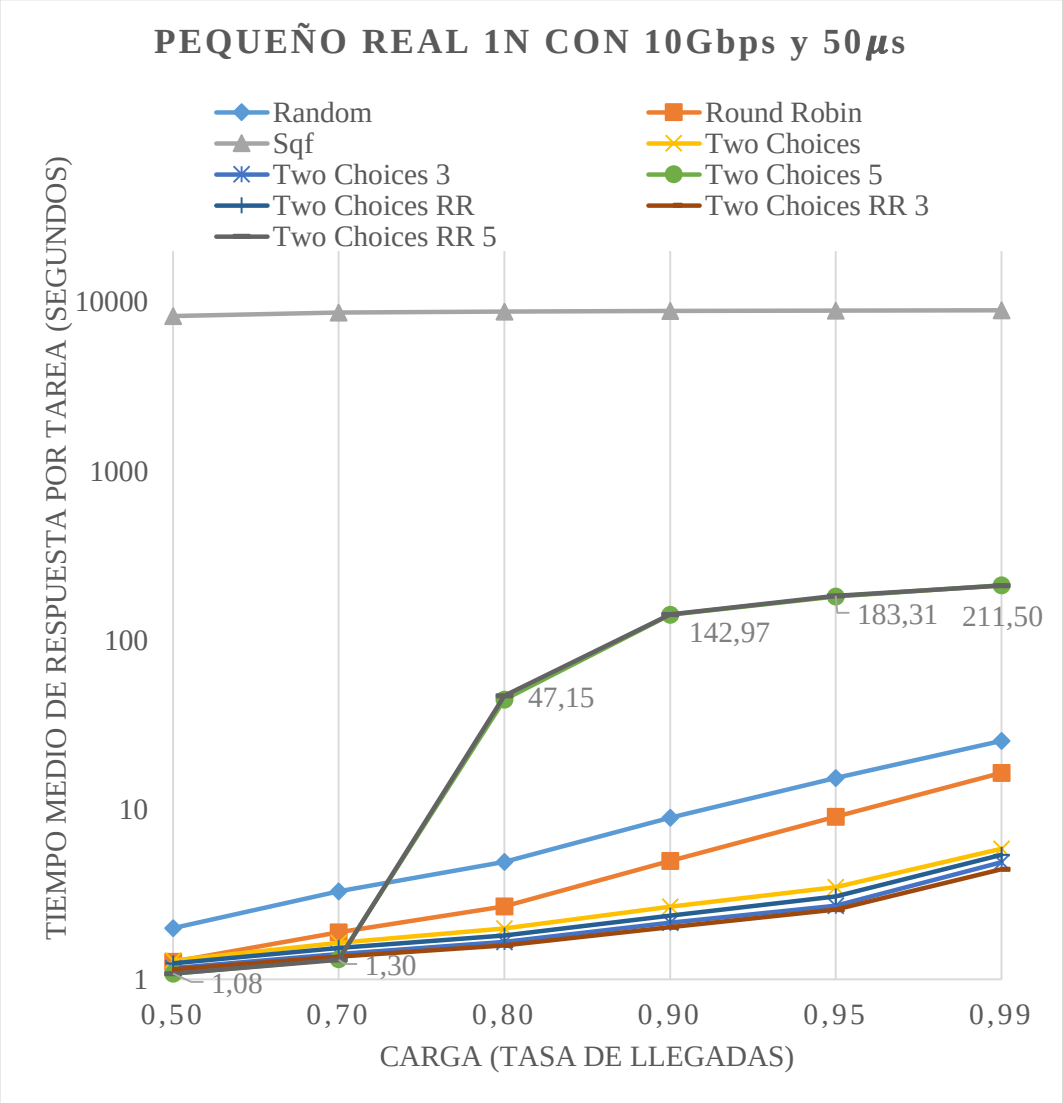
GRÁFICA 4: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA PEQUEÑA*

Como se ve en la gráfica 4, la cola en los hosts es prácticamente nula cuando los algoritmos que se evalúan son aquellos que intercambian mensajes de estado con los hosts, hecho que fundamenta la explicación aportada para la gráfica anterior; para esos algoritmos, el dispatcher de peticiones consume mucho tiempo determinando a qué host enviar la tarea, por lo que mientras éste está buscando el host más adecuado, al sistema le da tiempo a ejecutar las tareas que ha recibido y no se generan apenas colas en los host. En cambio, los valores de cola para los algoritmos *round robin* y *random* son muy parecidos a los obtenidos en la evaluación teórica ya que esos dos algoritmos se han comportado prácticamente igual en la evaluación real y en la evaluación teórica.

Para terminar con la evaluación real en la plataforma pequeña con links a 1Gbps, destacar que debido a que los tiempos medios obtenidos cubren un rango muy amplio de valores, se ha optado por cambiar el eje vertical de las gráficas 3 y 5 a una escala logarítmica.

Ahora que ya se ha descrito la evaluación real con links a 1Gbps, se van a interpretar los datos recogidos con el uso de links a 10Gbps. Los datos obtenidos en esta nueva evaluación están representados en las gráficas 5 y 6.

El cambio en las características de la red se ha traducido en una reducción del 53% en el tiempo medio necesario para la ejecución de una tarea con el algoritmo *shortest queue first* respecto del tiempo obtenido por el mismo algoritmo en la plataforma pequeña con links a 1Gbps. Aun así, sigue siendo el peor de todos los algoritmos evaluados.



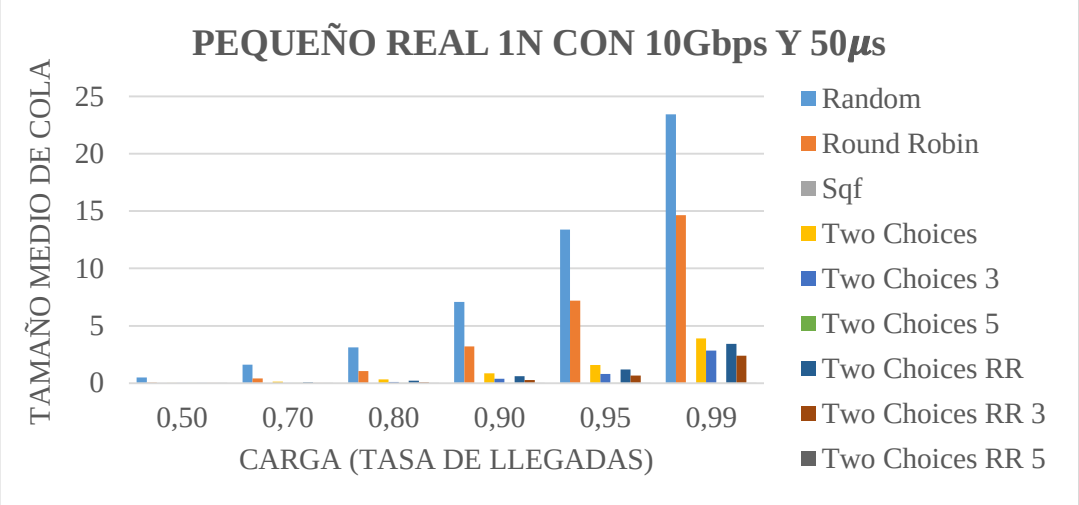
GRÁFICA 5: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA PEQUEÑA*

Lo primero que llama la atención es que con la mejora de la red el algoritmo que se comporta mejor ya no es el *round robin*, sino que han pasado a serlo dos, dependiendo de la carga que tenga el sistema; para cargas menores a 0,7 es con el *two random choices - round robin 5* con el que se obtienen mejores tiempos, mientras que para cargas superiores, es con el *two random choices - round robin 3*. Esto se debe a que cuando la tasa de llegadas toma un valor entre 0,5 y 0,7 la velocidad a la que el algoritmo *two choices - round robin 5* distribuye las tareas es suficiente para que no se produzcan esperas en el dispatcher de peticiones, mientras que cuando la velocidad a la que llegan las tareas desde el generador de peticiones aumenta, el dispatcher de peticiones no es capaz de distribuirlas a velocidad suficiente, por lo que se genera cola en el dispatcher y las tareas pierden tiempo esperando a ser distribuidas.

En cambio, para una carga a partir de 0,7 es con el algoritmo *two random choices-round robin 3* con el que se obtienen mejores resultados porque consigue encontrar

el equilibrio entre la cantidad de hosts a los que hay que preguntar sobre su cola sin que ello suponga reducciones del rendimiento debido a esperas en el dispatcher de peticiones.

El último aspecto a destacar de la gráfica es que gracias a la mejora de la de la red, el algoritmo *two random choices-round robin* obtiene resultados ligeramente mejores que el *two random choices*.



GRÁFICA 6: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA PEQUEÑA*

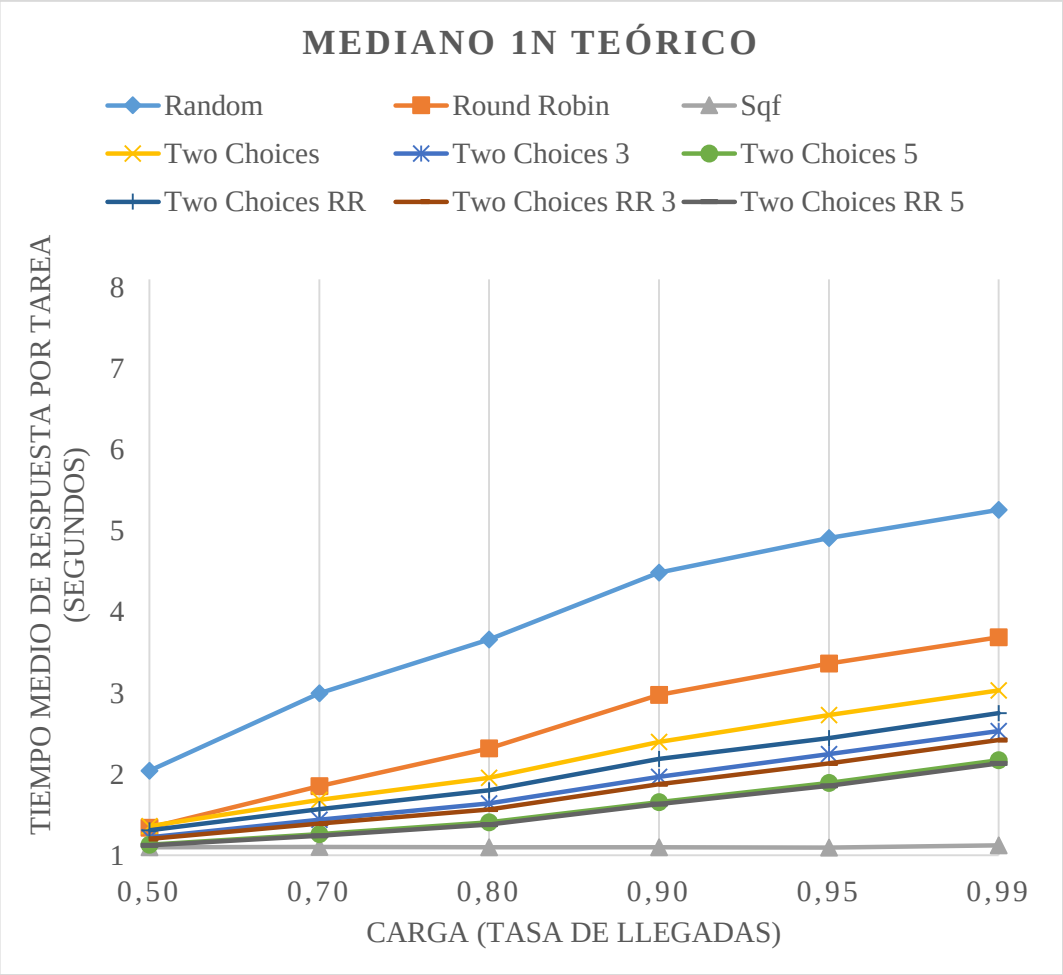
Para finalizar con la evaluación real en la plataforma pequeña, destacar que en la gráfica 6 se puede ver que el algoritmo *shortest queue first* ha generado tamaños medios de cola 0 para todos los valores de tasa de llegadas. Una vez más, esto se debe a que a los host les da tiempo a ejecutar la tarea recibida del dispatcher antes de recibir la siguiente.

4.8. EXPERIMENTACIÓN CON LA PLATAFORMA MEDIANA DE UN NIVEL: EVALUACIÓN TEÓRICA Y REAL

La experimentación en esta plataforma, al igual que la realizada en el punto 4.7, consiste en la generación de 100.000 tareas de la manera descrita en ese punto. Además, la características de las tareas para solicitar el tamaño de cola como también las de las tareas que se encargan de responder se mantienen invariables respecto a lo establecido en ese punto.

Las gráficas 7 y 8 son la evaluación teórica en la plataforma mediana de un nivel. Como se puede ver, casi no hay variación en el orden en el que cada algoritmo se ha comportado respecto de la evaluación teórica pequeña de un nivel; el algoritmo con el que peor resultados se obtiene es con el *random* y con el que mejor es con el *shortest queue first*. De nuevo, al igual que para la evaluación llevada a cabo en el punto 4.7, este resultado se debe a que el envío de los mensajes de petición en la evaluación teórica tiene un coste despreciable, permitiendo al *shortest queue first*

seleccionar el host más desahogado de entre todos los del sistema sin apenas penalización.

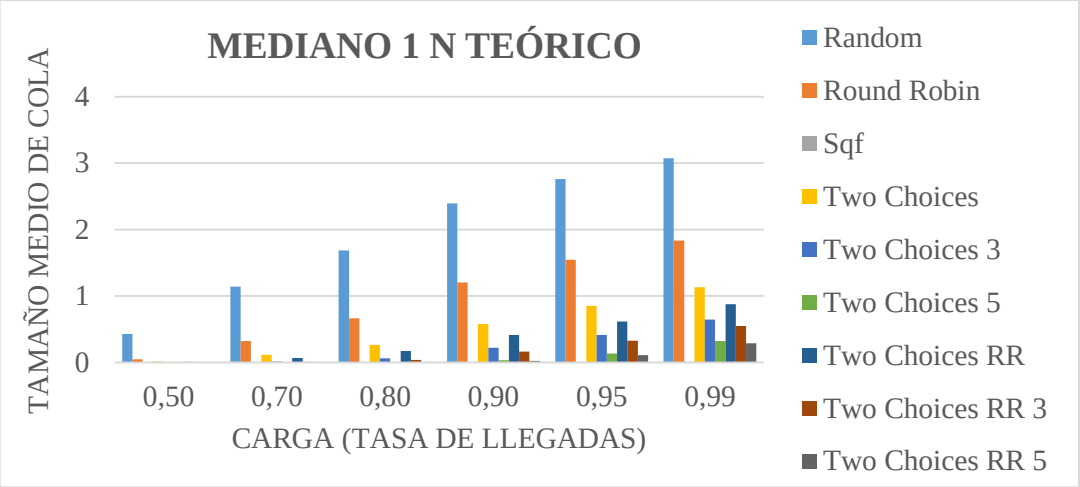


GRÁFICA 7: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA MEDIANA 1N*

Analizando detenidamente la gráfica 7, se llega de nuevo a la conclusión del punto 4.7; en la evaluación teórica, a medida que el algoritmo que se evalúa está mejor informado sobre las colas de los host, mejor es la elección que realiza para el destino de una tarea, lo que se refleja en una mejora del rendimiento. Esto es así ya que los links para la evaluación teórica hacen que enviar tareas por la red no suponga apenas coste.

Otro aspecto que se observa en la gráfica es que todos los algoritmos evaluados han reducido el tiempo medio de respuesta por tarea en la plataforma mediana respecto a los tiempos obtenidos con los mismos algoritmos en la evaluación teórica pequeña. Esto se debe a que el número de tareas generadas se ha mantenido en 100.000, por lo que aunque la tasa de llegadas ha aumentado considerablemente (ya que como se comentó, entre otros depende del número de host), al tener más hosts disponibles para ejecutar, menor ha sido el tiempo que el sistema tarda en ejecutar cada tarea.

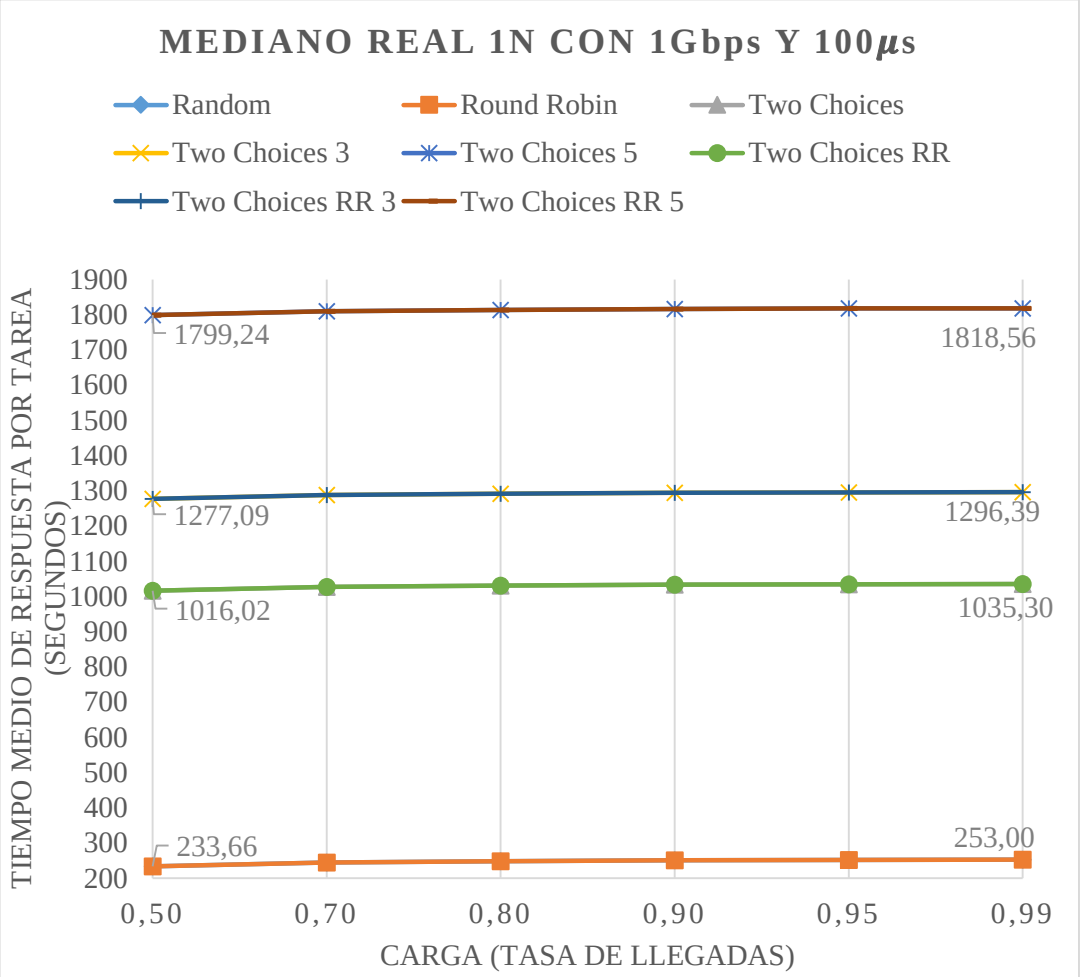
El último elemento que se quiere resaltar es que el rendimiento del *two random choices* es ligeramente peor que el que se obtiene a través de *two random choices-round robin*.



GRÁFICA 8: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA MEDIANA 1N*

Igual que para la evaluación teórica del punto 4.7, los algoritmos que hacen que en los host haya colas pequeñas son con los que se obtienen mejores resultados, mientras que los que generan colas grandes, son los de tiempos medios mayores. Por tanto, los algoritmos que distribuyen las tareas teniendo en cuenta el estado de los host consiguen mayor cantidad de tareas ejecutadas por unidad de tiempo en la evaluación teórica.

Ahora que ya se ha descrito la evaluación teórica, se va a estudiar la evaluación real realizada sobre esta plataforma. La evaluación real de los algoritmos se ha llevado a cabo sobre los dos modelos descritos en el punto 4.3, cuya única diferencia entre un modelo y otro son las características de los links de conexión que utilizan. Ésta evaluación se corresponde con las gráficas 9, 10, 11 y 12.

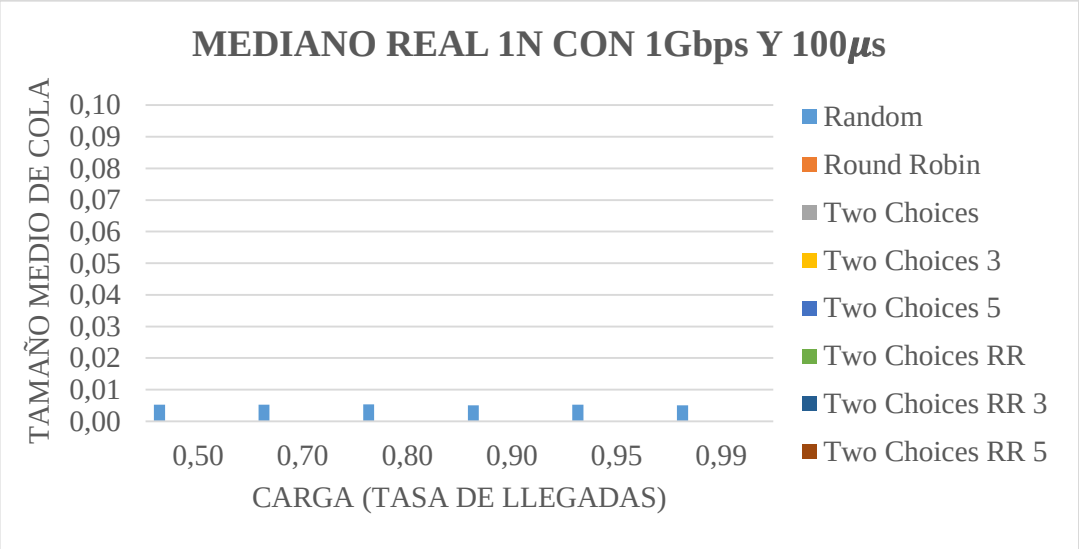


GRÁFICA 9: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA MEDIANA 1N*

Lo primero que hay que resaltar es que a partir de ahora, en las evaluaciones reales, no se va a estudiar el algoritmo *shortest queue first*. Se ha tomado esta decisión a partir de los datos obtenidos de la evaluación real en la plataforma pequeña, con la que se ha visto que la solución aportada por este algoritmo para un entorno real es muy ineficiente.

Una vez dicho esto, como se puede observar en la gráfica 9, los tiempos medios de respuesta por tarea han sido mayores en esta plataforma que para la misma evaluación en la plataforma pequeña. ¿Cómo puede ser que si lo único que ha cambiado de la plataforma pequeña a la mediana ha sido el número de clústeres, el rendimiento de la mediana sea peor si además no se ha modificado el número de tareas generadas y se ha mantenido en 100.000 mil?. La respuesta a esta pregunta está en la tasa de llegadas, que al depender de los hosts que tenga el sistema, produce que las tareas lleguen desde el generador de peticiones mucho más juntas en el tiempo que para la plataforma pequeña, por lo que la red se congestiona causando una reducción considerable del rendimiento.

El último aspecto que se puede observar en la gráfica 9 es que la congestión de la red ha causado que algoritmos con funcionalidad diferente se comporten de manera idéntica, ya que los cambios introducidos por uno u otro se encuentran condicionados por la red.

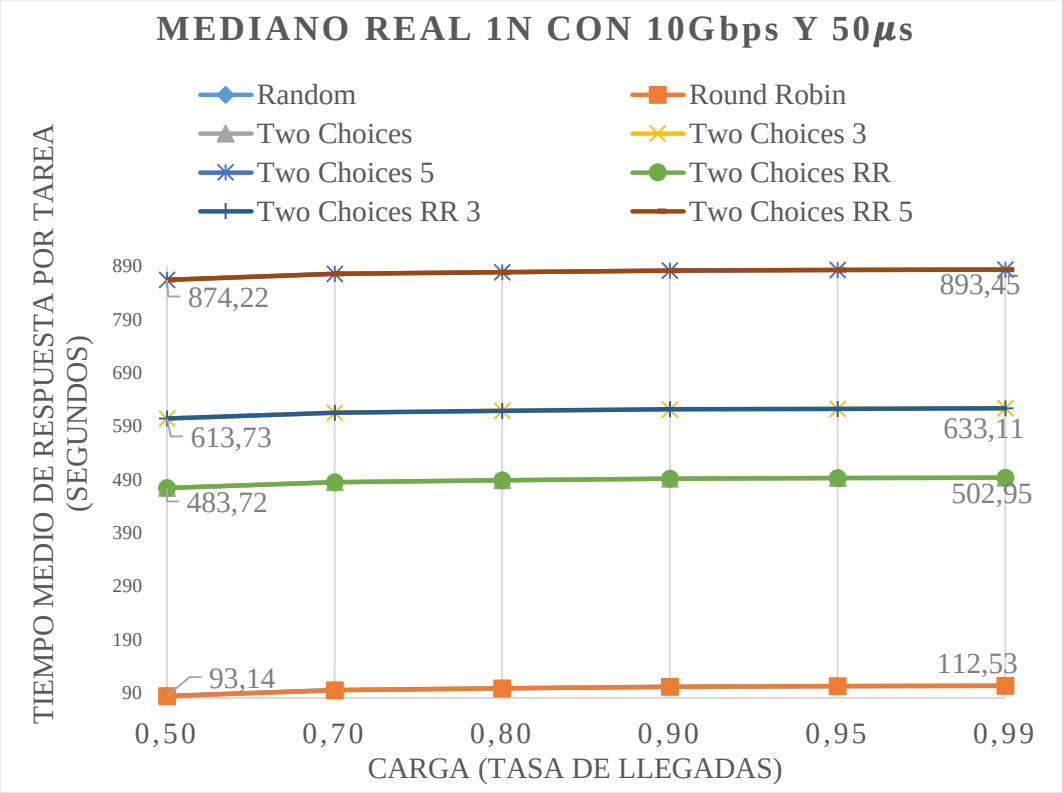


GRÁFICA 10: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA MEDIANA 1N*

Finalmente, en la gráfica 10 se puede ver que ninguno de los algoritmos ha generado cola en los host a excepción del *random*, hecho que fortalece lo descrito anteriormente; el rendimiento global del sistema se ve reducido por la saturación de la red, por lo que el *throughput* de tareas hacia los host es menor y no se generan colas ya que les da tiempo a ejecutar las tareas antes de recibir la siguiente.

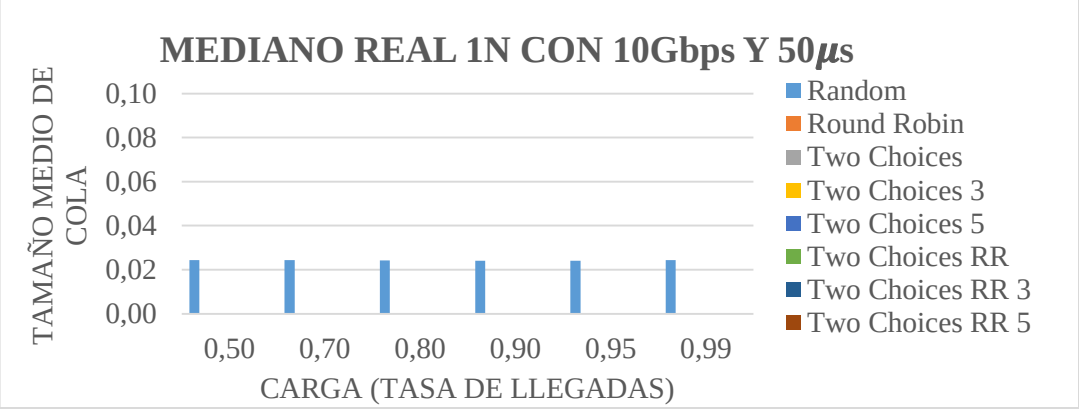
La segunda evaluación real se ha realizado sobre el modelo de plataforma con links de 10Gbps de ancho de banda y 50 μs de latencia, siendo las gráficas 11 y 12 la representación del resultado de la evaluación.

Al aumentar el ancho de banda y reducir la latencia de los links en la plataforma mediana de un nivel, el tiempo medio de respuesta por tarea obtenido para la evaluación real de cada uno de los algoritmos disminuye. Aun así, los tiempos siguen siendo peores que los de la evaluación de la plataforma pequeña, por lo que se puede afirmar que la mejora de la red no ha sido suficiente para hacer frente a la saturación debido a la altísima tasa de llegadas.



GRÁFICA 11: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA MEDIANA 1N*

A partir de los resultados obtenidos en la evaluación teórica de la gráfica 7, se puede afirmar que el problema de rendimiento en esta plataforma no se debe al hecho de que únicamente haya un dispatcher de peticiones, ya que los resultados obtenidos en esa evaluación son mejores que los de la plataforma pequeña. Por tanto, el problema está ocasionado por el diseño descrito en el punto 4.3, donde se define un único link de conexión entre el dispatcher de peticiones y cada clúster. En la plataforma pequeña, este diseño funciona bien debido a que hay 24 host por clúster, pero en la plataforma mediana, al tener 256 host por clúster, ese link hace de cuello de botella al recibir más información de la que puede transmitir.

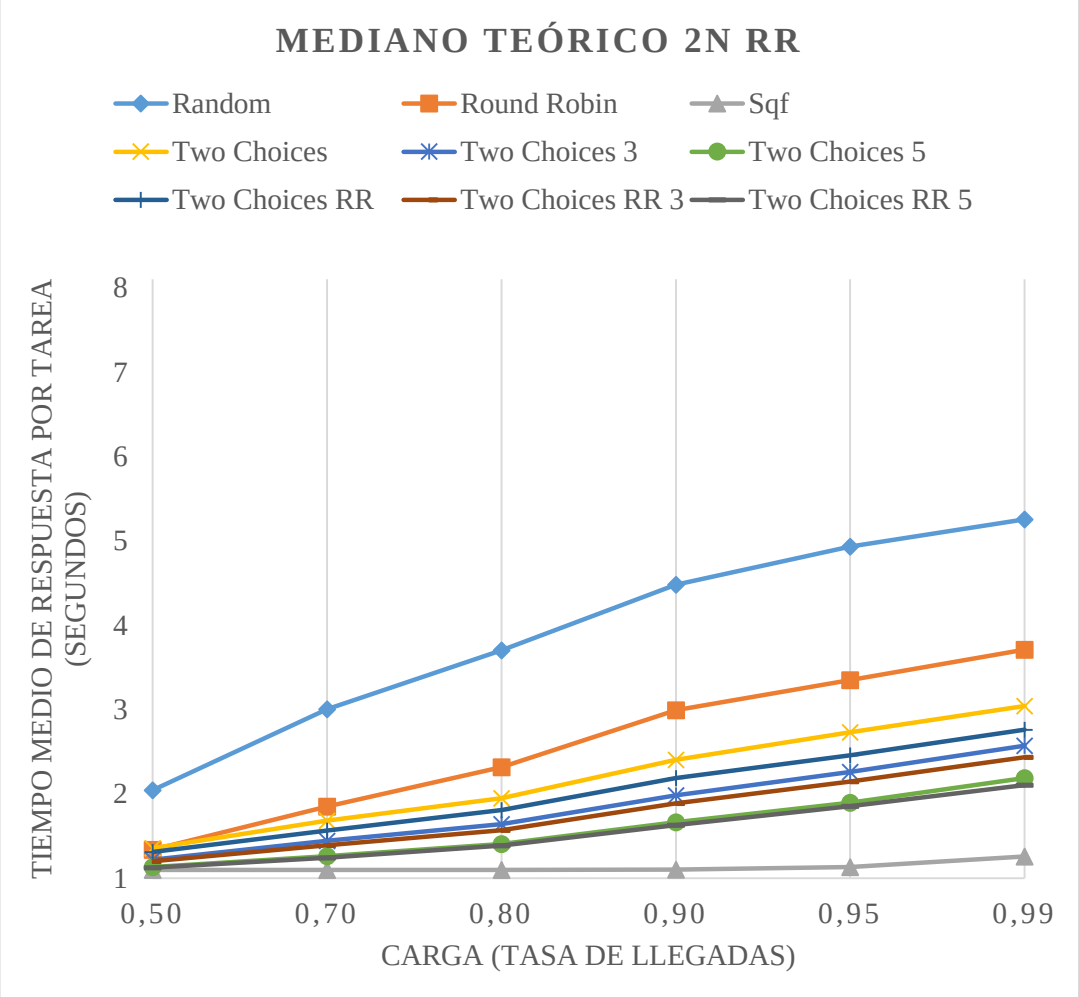


GRÁFICA 12: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA MEDIANA 1N*

4.9. EXPERIMENTACIÓN CON LA PLATAFORMA MEDIANA DE DOS NIVELES: EVALUACIÓN TEÓRICA Y REAL

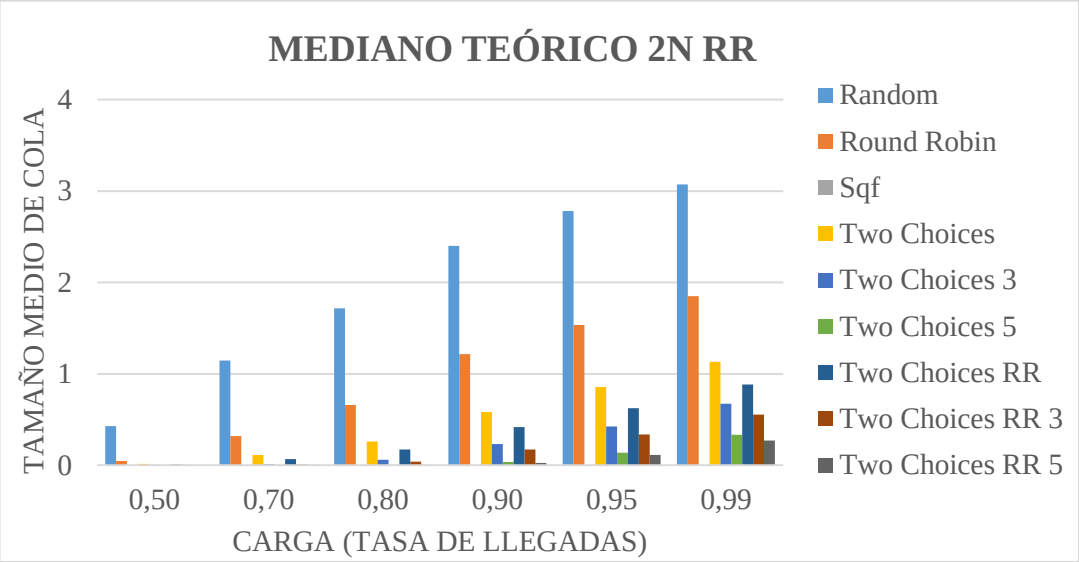
En este punto se van a evaluar los algoritmos en la plataforma mediana de dos niveles descrita en el punto 4.3. De nuevo, de la misma forma que en las evaluaciones de los puntos anteriores, las propiedades de las tareas como también el número que se generan de éstas coinciden con las del punto 4.7.

Una vez dicho esto, las gráficas 13 y 14 representan los valores obtenidos en la evaluación teórica con esta plataforma y el algoritmo *round robin* en el dispatcher de peticiones. Los tiempos medios de respuesta por tarea para la gráfica 13 son exactamente iguales a los obtenidos para la misma evaluación en la plataforma mediana de un nivel (gráfica 7). Por tanto, a nivel teórico, dedicar un host de los 256 para que haga de dispatcher dentro del clúster no supone mejora alguna del rendimiento.



GRÁFICA 13: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA MEDIANA DE 2N Y ROUND ROBIN*

Como era de esperar, el comportamiento de las colas de los host en la evaluación teórica con esta plataforma ha sido la misma que para la plataforma mediana de un nivel. Es por ello que para no duplicar información, se toma como buena la descripción dada para la gráfica 8.



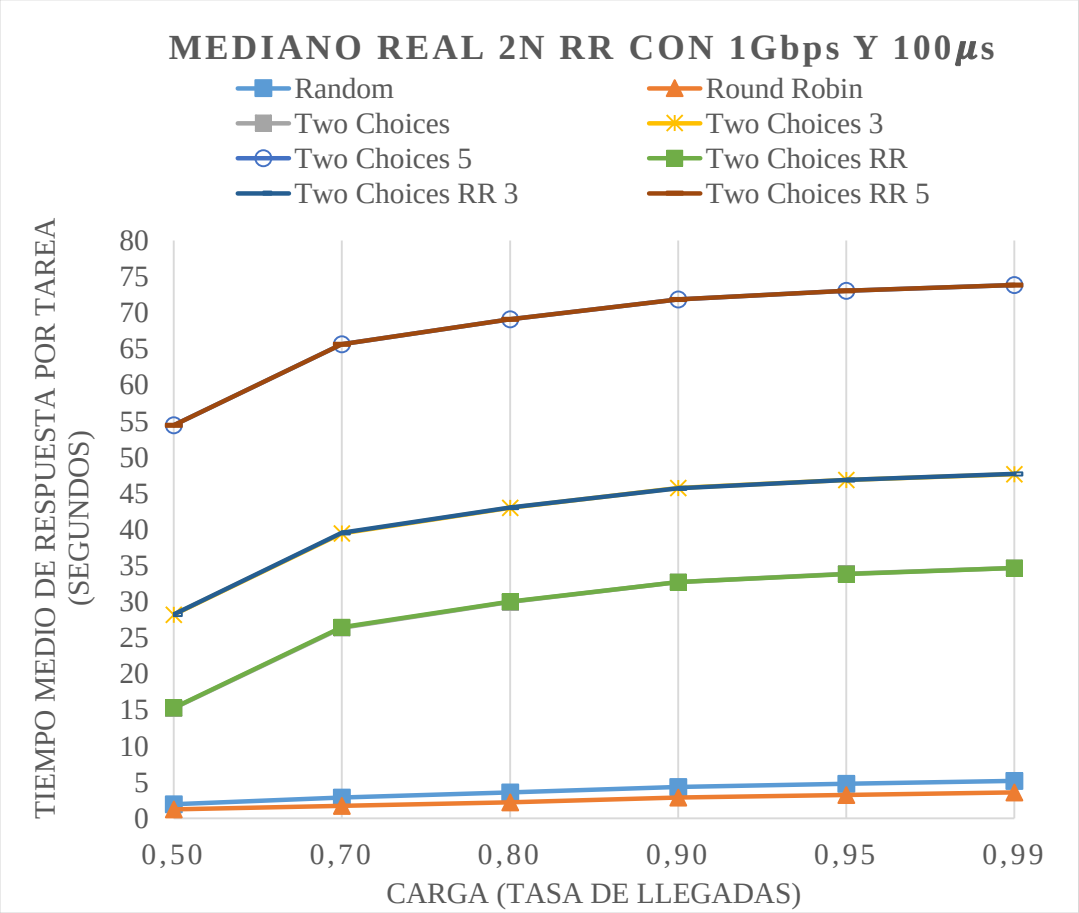
GRÁFICA 14: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA MEDIANA 2N Y ROUND ROBIN*

Una vez descrita la evaluación teórica en la plataforma mediana de dos niveles cuando el algoritmo que actúa en el dispatcher de peticiones es *round robin*, se va a proceder a analizar los resultados obtenidos para la evaluación real. Las gráficas 15, 16, 17 y 18 muestran los datos recabados en la evaluación real con los dos modelos de links.

Lo primero a destacar de la gráfica 15 es que los tiempos de respuesta por tarea se han reducido notablemente respecto de la misma evaluación en la plataforma de un nivel (gráfica 9). Por tanto, a diferencia de lo que predecía la evaluación teórica en esta plataforma, la incorporación de un host encargado de distribuir las tareas dentro de los clústeres produce que el rendimiento mejore en la evaluación real. Esto se debe a que ahora el dispatcher de peticiones distribuye las tareas a medida que le llegan de manera *round robin*, por lo que las tareas en la evaluación real ya no pierden tiempo esperando en la cola del dispatcher de peticiones a ser distribuidas.

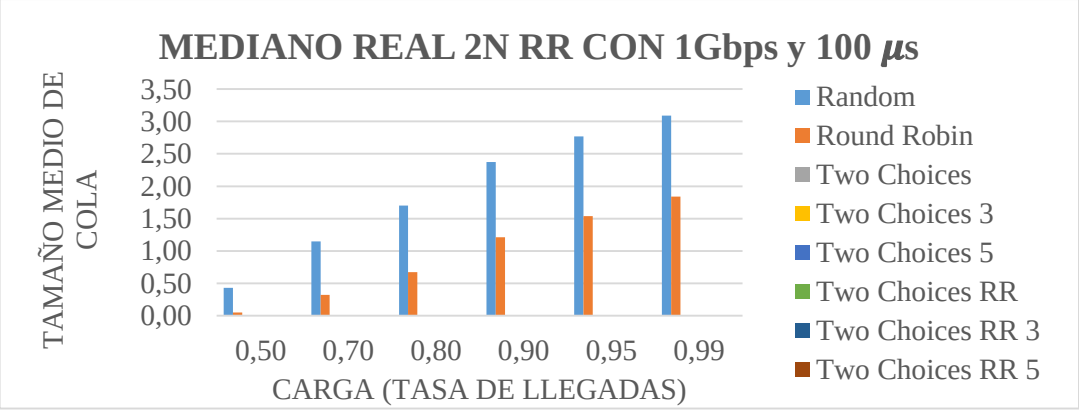
Otra característica interesante es que el rendimiento del sistema no varía se use el algoritmo *two choices* o el *two choices-round robin* en el dispatcher de clúster.

El segundo aspecto a destacar respecto a la evaluación de la gráfica 15 es que de nuevo, a medida que los algoritmos se informan sobre el estado de las colas de más host peor es el rendimiento que se obtiene con su uso. Es por esto que con el algoritmo que se han obtenido de nuevo los mejores resultados ha sido con el *round robin*.



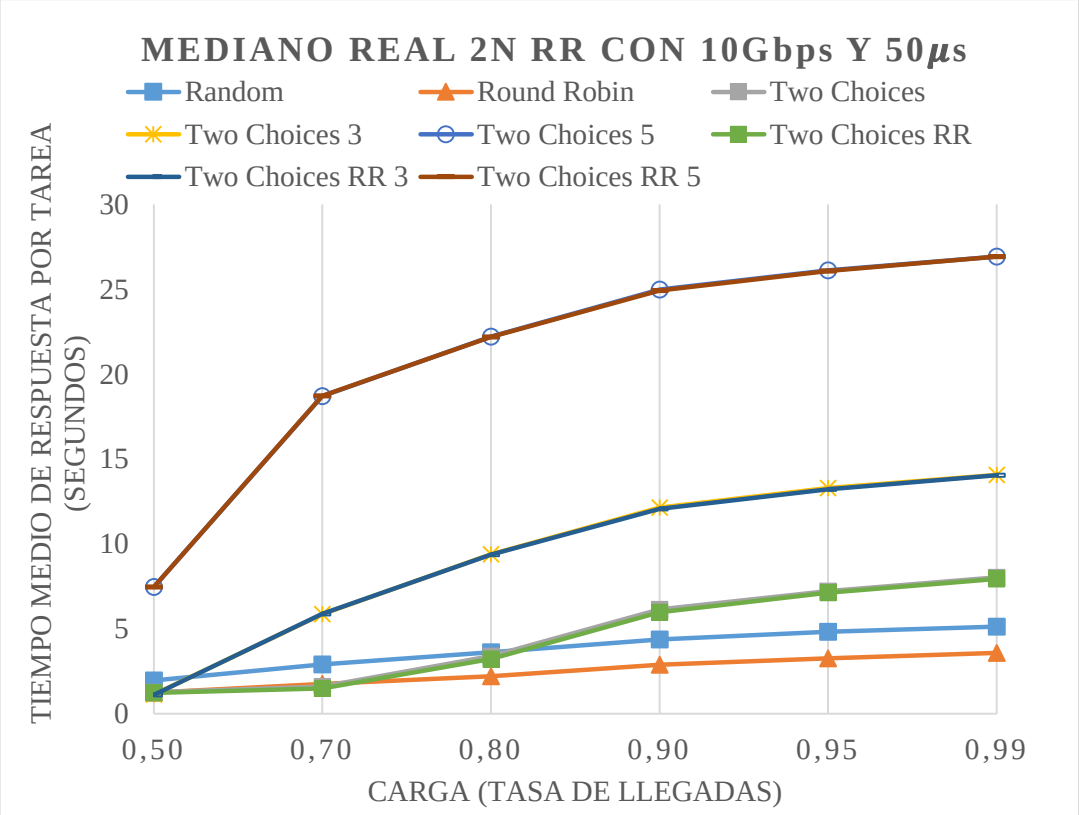
GRÁFICA 15: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA MEDIANA 2N Y ROUND ROBIN*

Finalmente, destacar que el uso del algoritmo *round robin* en el dispatcher de peticiones y la introducción de un host distribuidor en los clústeres ha hecho que el rendimiento del sistema mejore. Esto se debe a que ahora el dispatcher de peticiones tiene menos carga de trabajo, ya que es el dispatcher de clúster el que distribuye las tareas dentro del clúster. De esta manera, el dispatcher de peticiones puede enviar más tareas por unidad de tiempo a los clústeres con la mejora que eso supone.



GRÁFICA 16: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA MEDIANA 2N Y ROUND ROBIN*

La gráfica 17 muestra la evaluación real de dos niveles *round robin* con links a 10Gbps en la plataforma mediana. Lo primero que se aprecia es que al instalar links con mayor ancho de banda y menor latencia, el tiempo medio de respuesta medio con los algoritmos del tipo *two random choices* disminuye. Esto es lógico, ya que con las nuevas características de la red, el envío de tareas de petición y sus respectivas respuestas no están tan penalizadas como cuando la conexión era a 1Gbps.

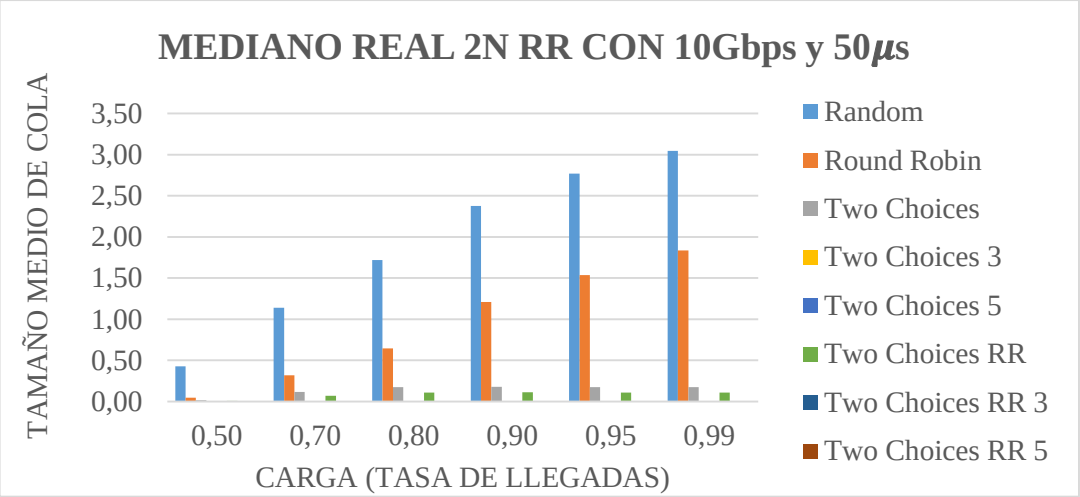


GRÁFICA 17: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA MEDIANA 2N Y ROUND ROBIN*

Otra faceta a destacar de la evaluación anterior y que la diferencia de la mostrada en la gráfica 15 es que el algoritmo *round robin* ya no es el mejor para todas las tasas de llegadas; para tasas entre 0,5 y 0,75 el algoritmo que mejor se comporta como distribuidor de la carga dentro de los clústeres es el *two choices*, mientras que cuando la tasa de llegadas es 0,75 o más, *round robin* vuelve a ser el mejor. De nuevo, esto sucede por la mejora de la red, que hace posible que para tasas de llegada menores a 0,75 sea viable preguntar a dos host sobre el estado de su cola sin que se acumulen tareas en el dispatcher de clúster y por lo tanto se obtengan peores resultados que con *round robin* y *random*.

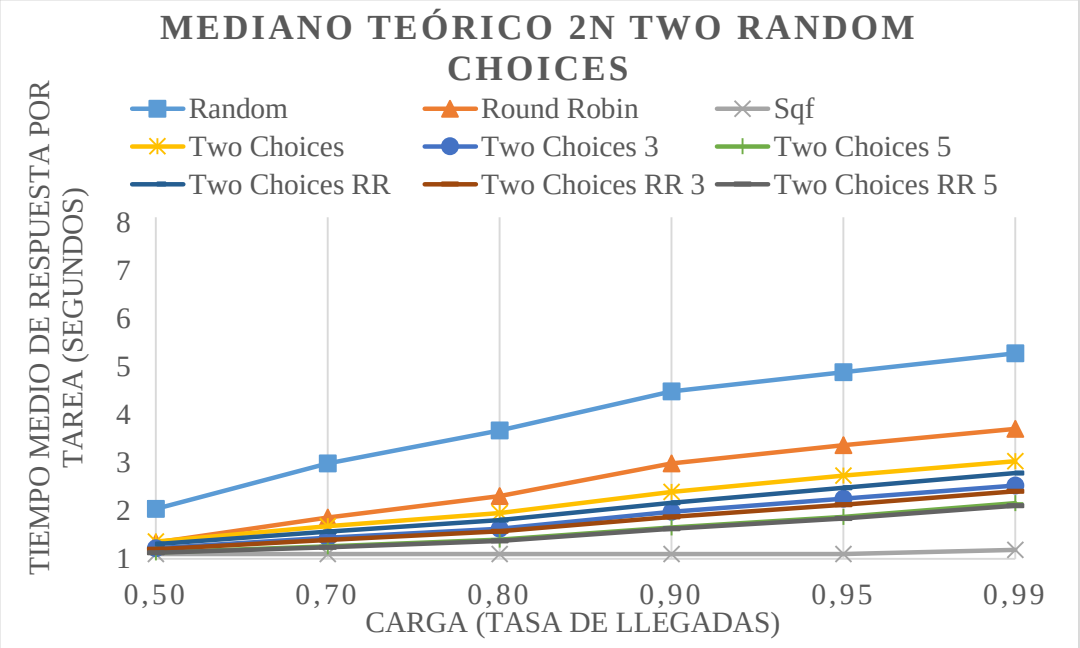
Finalmente, comentar que a partir de la tasa de llegadas de 0,75 el algoritmo *round robin* vuelve a ser con el que se obtienen mejores tiempos de respuesta medio. Esto está relacionado con el hecho de que al aumentar la tasa de llegadas, las tareas llegan más juntas en el tiempo, pero en cambio, el algoritmo *two choices* sigue requiriendo

el mismo tiempo para enviar la tarea de petición por la red y recibir la respuesta (ya que la red sigue siendo la misma), por lo que se acumulan tareas en la cola del dispatcher de clúster con la bajada de rendimiento que ello supone.



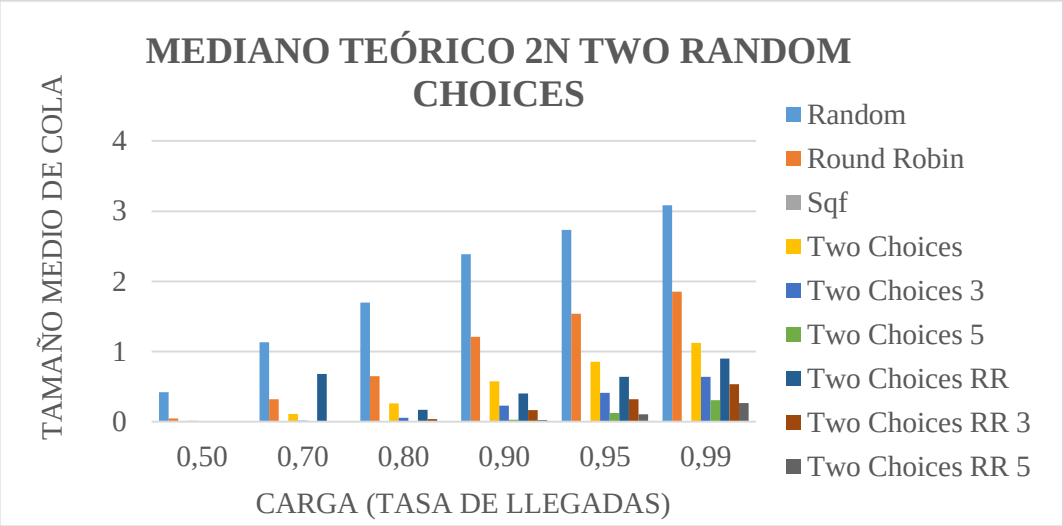
GRÁFICA 18: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA MEDIANA 2N Y ROUND ROBIN*

Una vez descritas las evaluaciones teóricas y reales con la plataforma mediana de dos niveles y *round robin* en el dispatcher de peticiones, se van a evaluar los resultados obtenidos cuando en el dispatcher de peticiones el algoritmo encargado de distribuir las tareas a los dispatcher de clúster es el *two random choices*. Los resultados teóricos de la nueva evaluación se muestran en las gráficas 19 y 20, mientras que los datos obtenidos en la evaluación real en las gráficas 21, 22, 23 y 24.



GRÁFICA 19: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA MEDIANA DE 2N Y TWO RANDOM CHOICES*

Lo primero que hay que destacar es que en la evaluación teórica en la plataforma mediana de dos niveles se obtienen los mismos resultados independientemente de si se usa *two random choices* o *round robin* en el dispatcher de peticiones. A esta conclusión se llega a través de comparar las gráficas 13 y 19.



GRÁFICA 20: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA MEDIANA 2N Y TWO RANDOM CHOICES*

Además, el comportamiento de las colas en los host es el mismo que el de la evaluación teórica de la gráfica 14. Por todo esto, se puede afirmar que en la evaluación teórica y para las características de nuestro sistema, poner un algoritmo que tenga o no en cuenta las colas en los clústeres es indiferente ya que el rendimiento que se obtiene es el mismo.

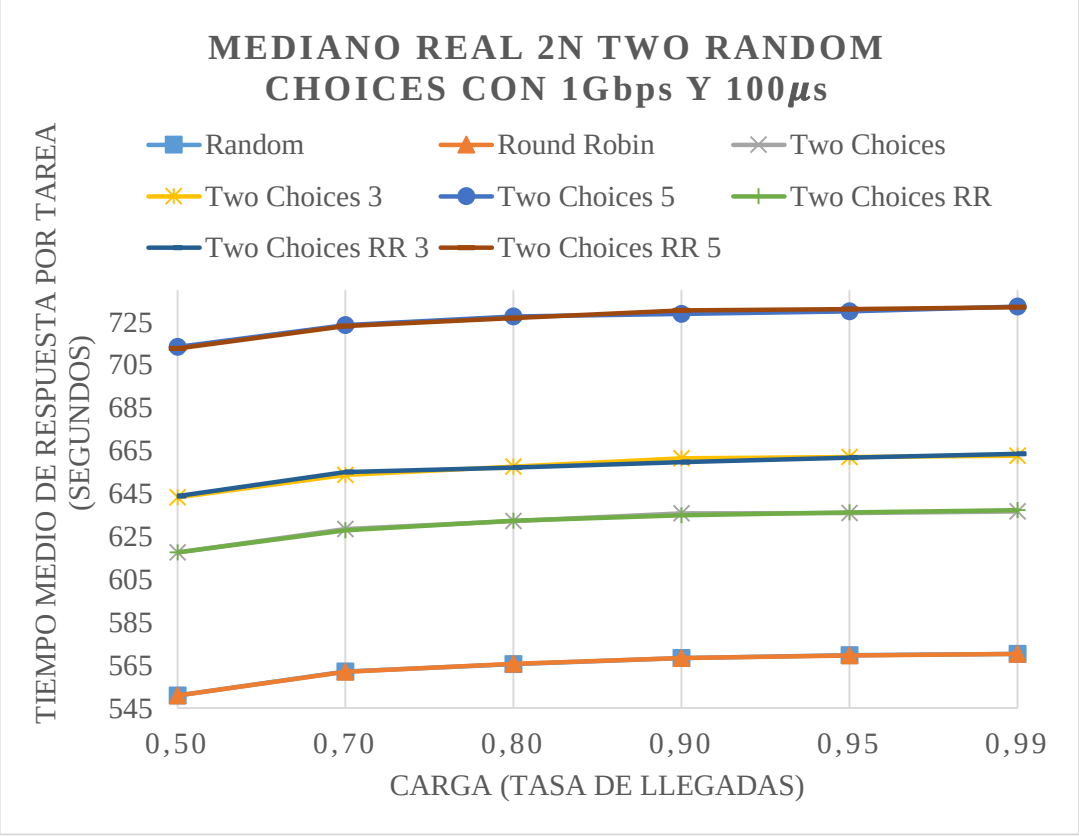
Ahora que ya se han descrito las características más importantes de la evaluación teórica, con la ayuda de las gráficas 21, 22, 23 y 24, se va a describir el comportamiento de los algoritmos en la evaluación real con la plataforma mediana de dos niveles y *two random choices* en el dispatcher de peticiones.

En la gráfica 21 y 23 se puede ver que los resultados extraídos nada tienen que ver con los obtenidos en la evaluación teórica.

Lo primero que hay que señalar de las evaluaciones reales es que, cuando en la plataforma mediana de dos niveles se emplea el algoritmo *two random choices* en el dispatcher de peticiones, el rendimiento del sistema disminuye considerablemente en comparación con la utilización de *round robin*. Como ha sucedido con casi todas las evaluaciones realizadas hasta ahora, el rendimiento de los algoritmos del tipo *two random choices* depende de las características de la red, por lo que situar este tipo de algoritmo en el dispatcher de peticiones no parece una buena idea.

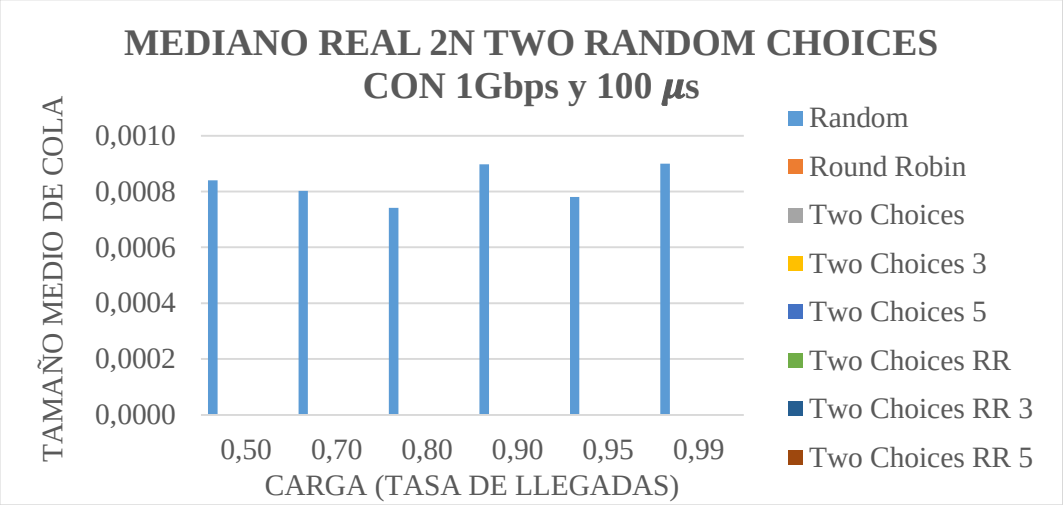
Asimismo, los tiempos medios obtenidos son menores en las evaluaciones reales con la plataforma mediana de dos niveles y *two random choices,* que los de la evaluación

con la plataforma mediana de 1 nivel. La causa de esto tiene que ver con la arquitectura de las plataformas de dos niveles; primero se escoge un clúster mediante el dispatcher de peticiones y luego, en ese clúster, es donde se selecciona el host de destino. Por tanto, la arquitectura permite crear grupos de host más reducidos, permitiendo que la elección de los algoritmos sea mucho mejor respecto de la que pueden hacer en una arquitectura de un nivel. Además, debido a que el dispatcher de clúster tiene una variable que provee en todo momento el número de tareas esperando a ser ejecutadas en el clúster, no pierde tiempo gestionando mensajes de petición y respuesta.



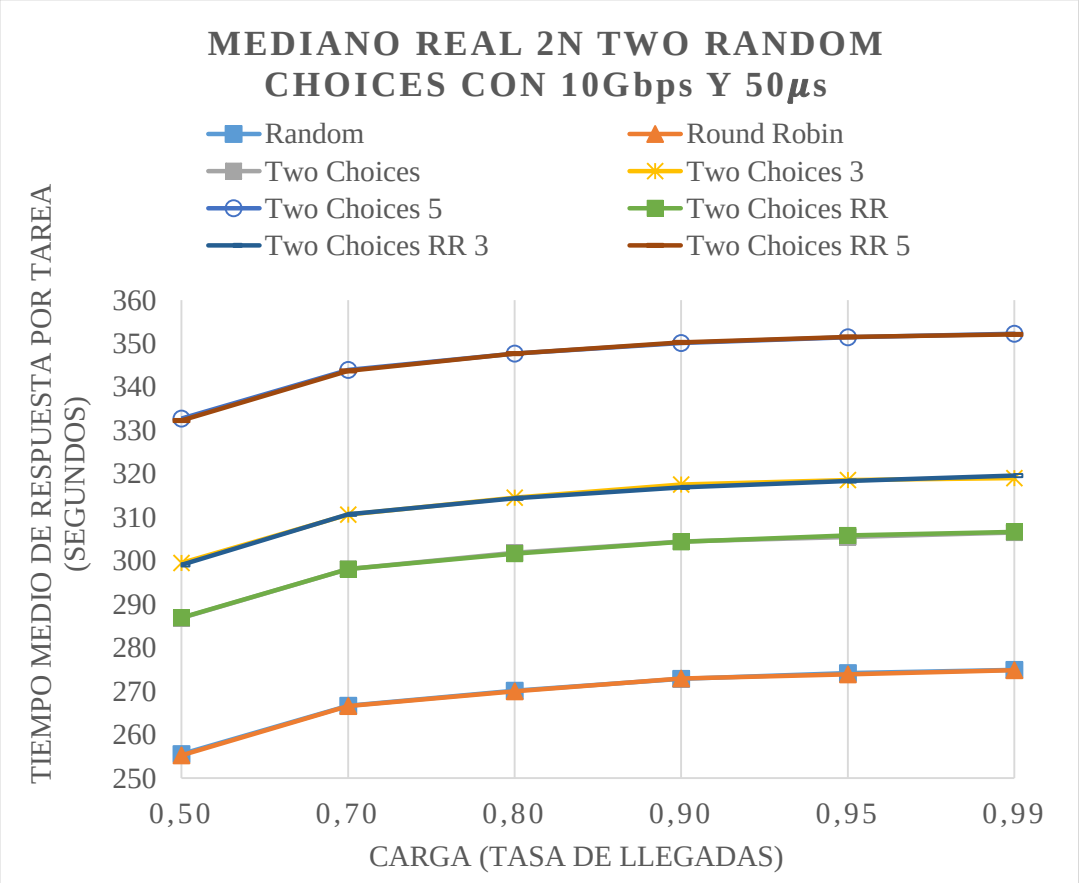
GRÁFICA 21: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA MEDIANA 2N Y TWO RANDOM CHOICES*

Para terminar de comentar la evaluación mostrada en la gráfica 21, destacar que el único algoritmo que ha generado algo de cola en los host ha sido el *random*.



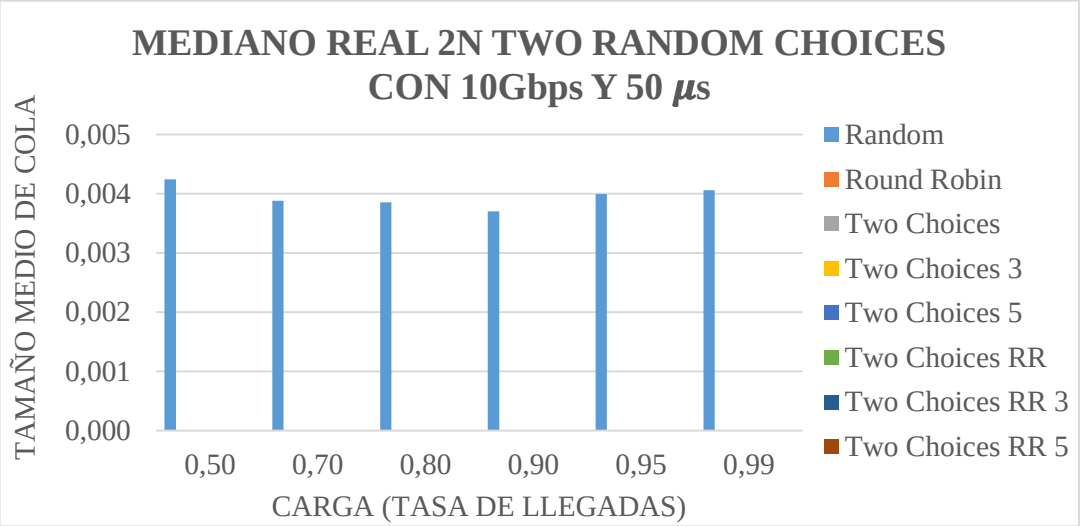
GRÁFICA 22: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA MEDIANA 2N Y TWO RANDOM CHOICES*

Si la evaluación mostrada en la gráfica 21 se realiza sobre la misma plataforma pero con links a 10Gbps de ancho de banda, se obtienen los resultados mostrados en la gráfica 23. Al aumentar el ancho de banda de las conexiones y disminuir la latencia, los tiempos medios de ejecución por tarea se reducen.



GRÁFICA 23: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA MEDIANA 2N Y TWO RANDOM CHOICES*

Finalmente, para finalizar con las evaluaciones en la plataforma mediana de dos niveles, destacar que únicamente el algoritmo *random* ha generado cola debido a que con los otros algoritmos a los host les da tiempo a ejecutar las tareas antes de recibir la siguiente.

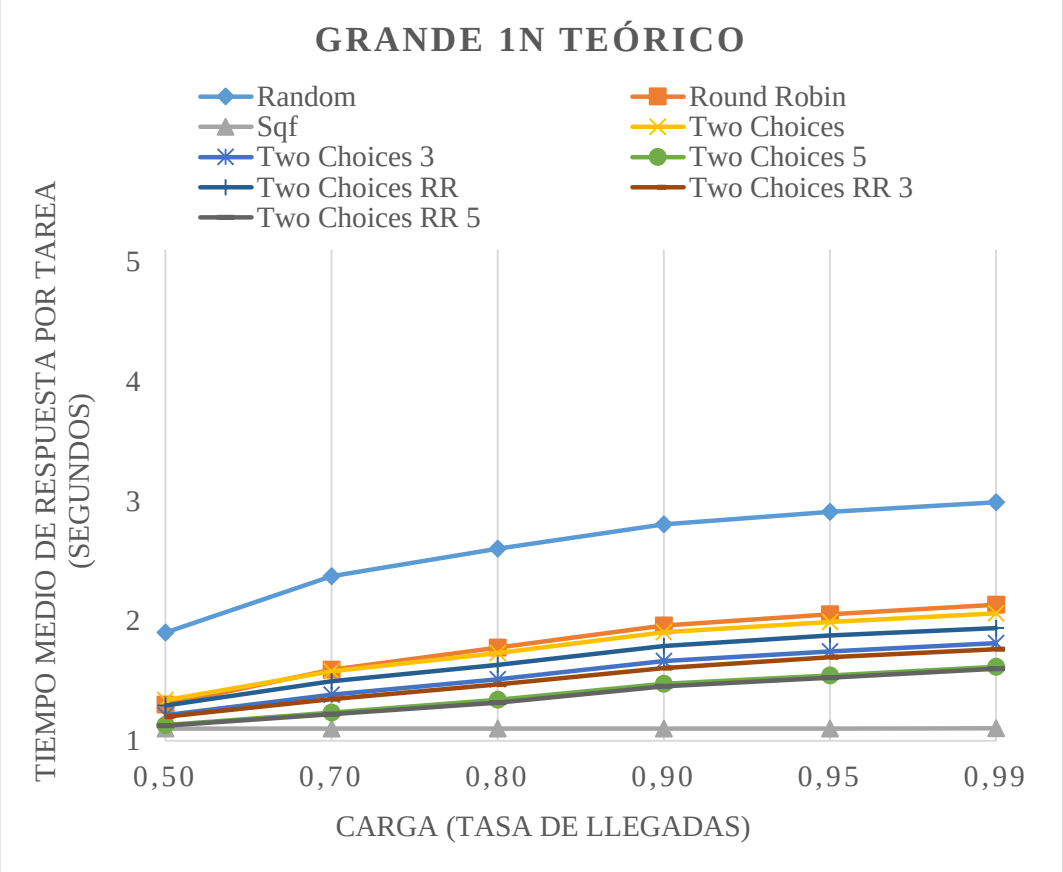


GRÁFICA 24: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA MEDIANA 2N Y TWO RANDOM CHOICES*

4.10. EXPERIMENTACIÓN CON LA PLATAFORMA GRANDE DE UN NIVEL: EVALUACIÓN TEÓRICA Y REAL

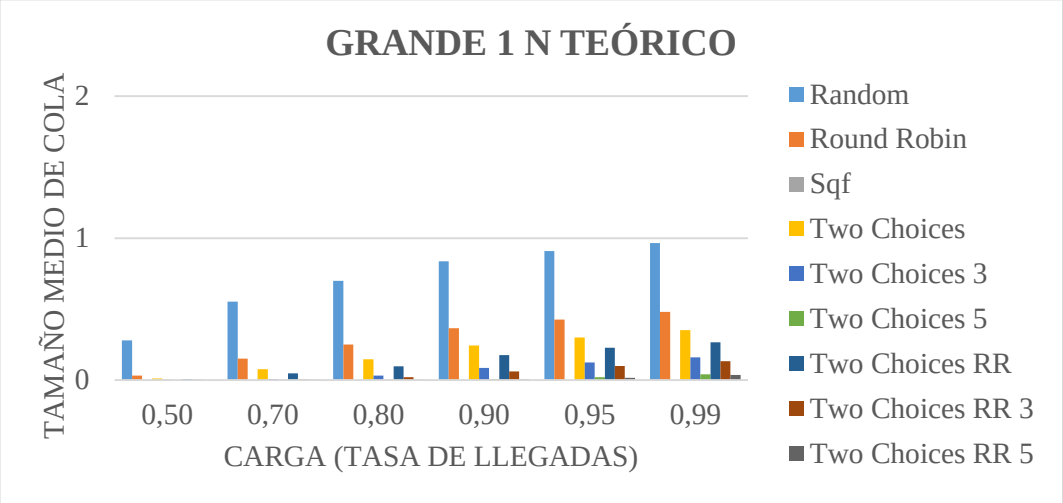
La experimentación en esta plataforma, al igual que la realizada en todos los puntos anteriores, consiste en la generación de 100.000 tareas de la manera descrita en el punto 4.7. Además, la características de las tareas de solicitud y respuesta para los tamaños de cola tampoco varían respecto a lo establecido en ese punto.

Las gráficas 25 y 26 representan la evaluación teórica en la plataforma grande de un nivel. Como se puede ver, la única variación respecto al comportamiento obtenido en las evaluaciones teóricas para las plataformas pequeña y mediana de un nivel ha sido que con la plataforma grande se obtienen mejores tiempos.



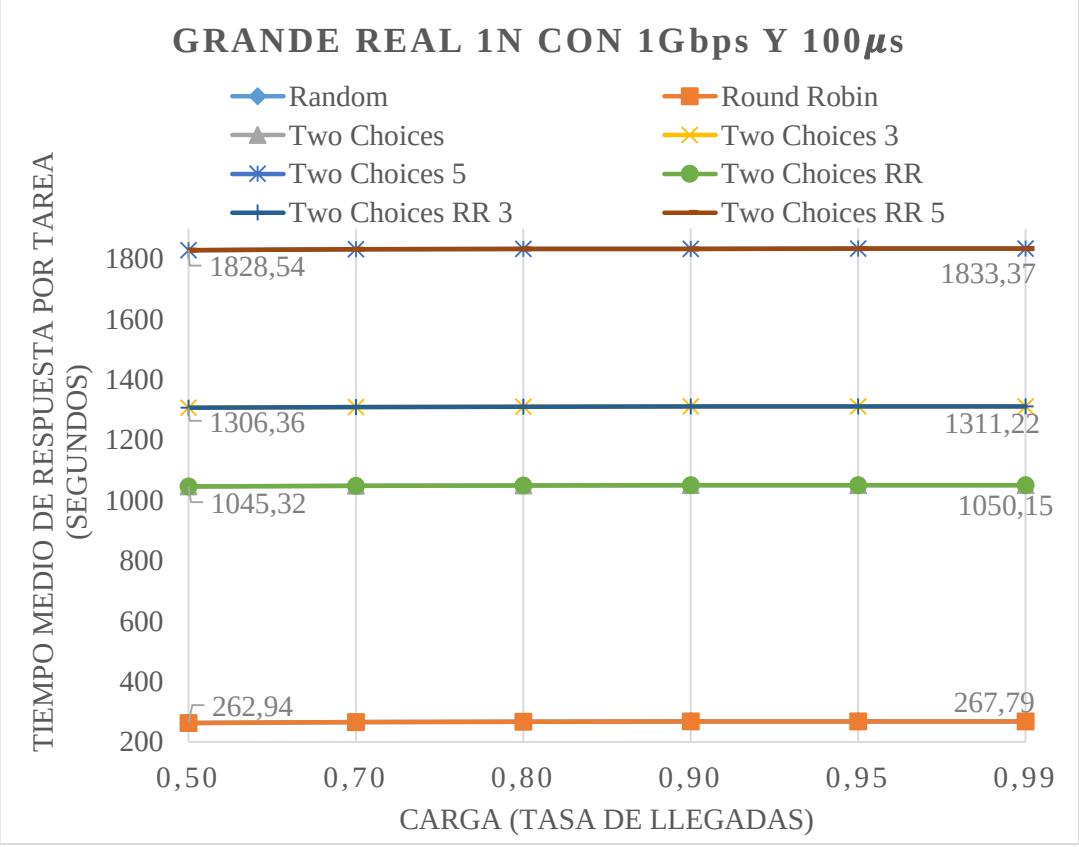
GRÁFICA 25: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA GRANDE 1N*

La descripción aportada en los puntos 4.7 y 4.8 de la evaluación teórica con la plataforma pequeña y mediana de un nivel sirve para explicar las gráficas 25 y 26, por lo que para no repetir información, se toman esas explicaciones como válidas.



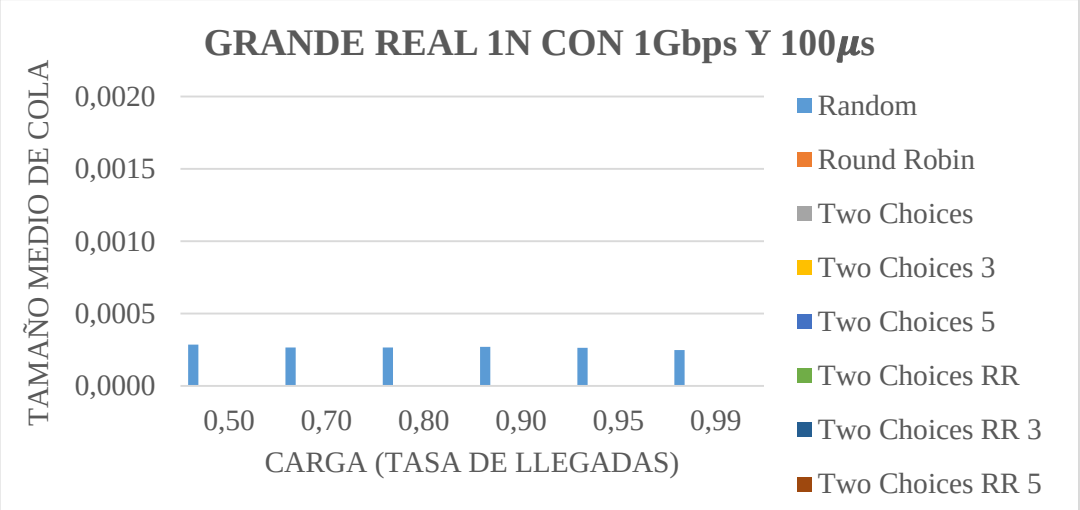
GRÁFICA 26: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA GRANDE 1N*

Ahora que ya se ha descrito la evaluación teórica en la plataforma grande de un nivel, se va a analizar la evaluación real. La evaluación real de los algoritmos se ha llevado a cabo sobre los dos modelos de plataforma grande de un nivel descritos en el punto 4.5, cuya única diferencia entre un modelo y otro son las características de los links de conexión que utilizan. Ésta evaluación se corresponde con las gráficas 27, 28, 29 y 30.



GRÁFICA 27: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA GRANDE 1N*

Como se observa en la gráfica 27, los tiempos medios de respuesta por tarea han sido ligeramente mayores en esta plataforma que para la misma evaluación en la plataforma mediana. Como se ha comentado en el punto 4.8, la bajada de rendimiento respecto a la plataforma pequeña se debe a la tasa de llegadas, que al depender de los hosts que tiene el sistema, hace que las tareas lleguen mucho más juntas en el tiempo, por lo que la red se congestiona reduciendo considerablemente el rendimiento.

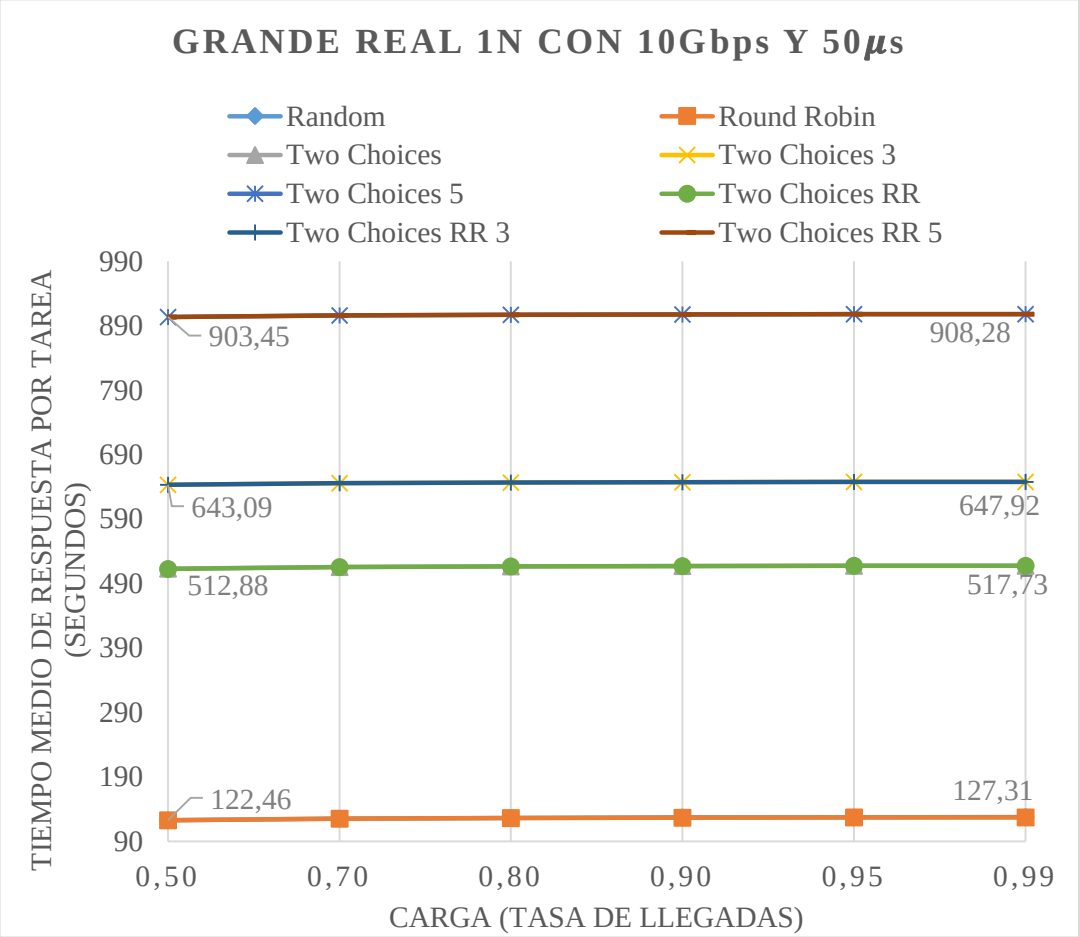


GRÁFICA 28: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA GRANDE 1N*

Finalmente, en la gráfica 28 se puede ver que el único algoritmo que ha generado colas en los host ha sido el *random* ya que a los host les da tiempo a ejecutar las tareas antes de recibir la siguiente.

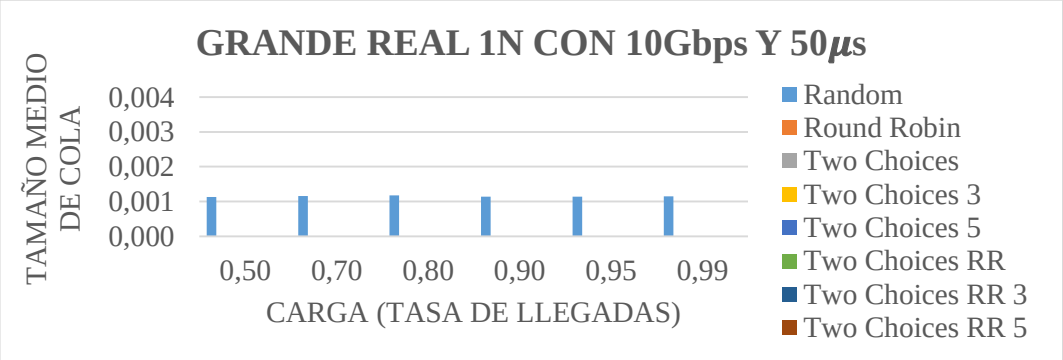
La segunda evaluación real en la plataforma grande de un nivel se ha llevado a cabo sobre el modelo con links a 10Gbps de ancho de banda y 50 μs de latencia, siendo las gráficas 29 y 30 la representación del resultado de dicha evaluación.

Al aumentar el ancho de banda y reducir la latencia de los links, el tiempo medio de respuesta por tarea disminuye. Aun así, los tiempos siguen siendo peores que los de la evaluación en la plataforma pequeña y mediana, por lo que se puede afirmar que la mejora de la red es insuficiente para hacer frente a la saturación provocada por la altísima tasa de llegadas.



GRÁFICA 29: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA GRANDE 1N*

Tal y como se ha comentado en el punto 4.8 para la evaluación en la plataforma mediana de un nivel, el empeoramiento del rendimiento no se debe únicamente a que haya un único dispatcher de peticiones en el sistema. Por tanto, el diseño de la plataforma ha tenido mucho que ver, ya que al haber un único link de conexión por clúster con el dispatcher de peticiones, cuando se aumenta el número de host en los clústeres, también lo hace la información a transmitir por ese link, por lo que el link no es capaz de despachar tanta información y la red se satura.

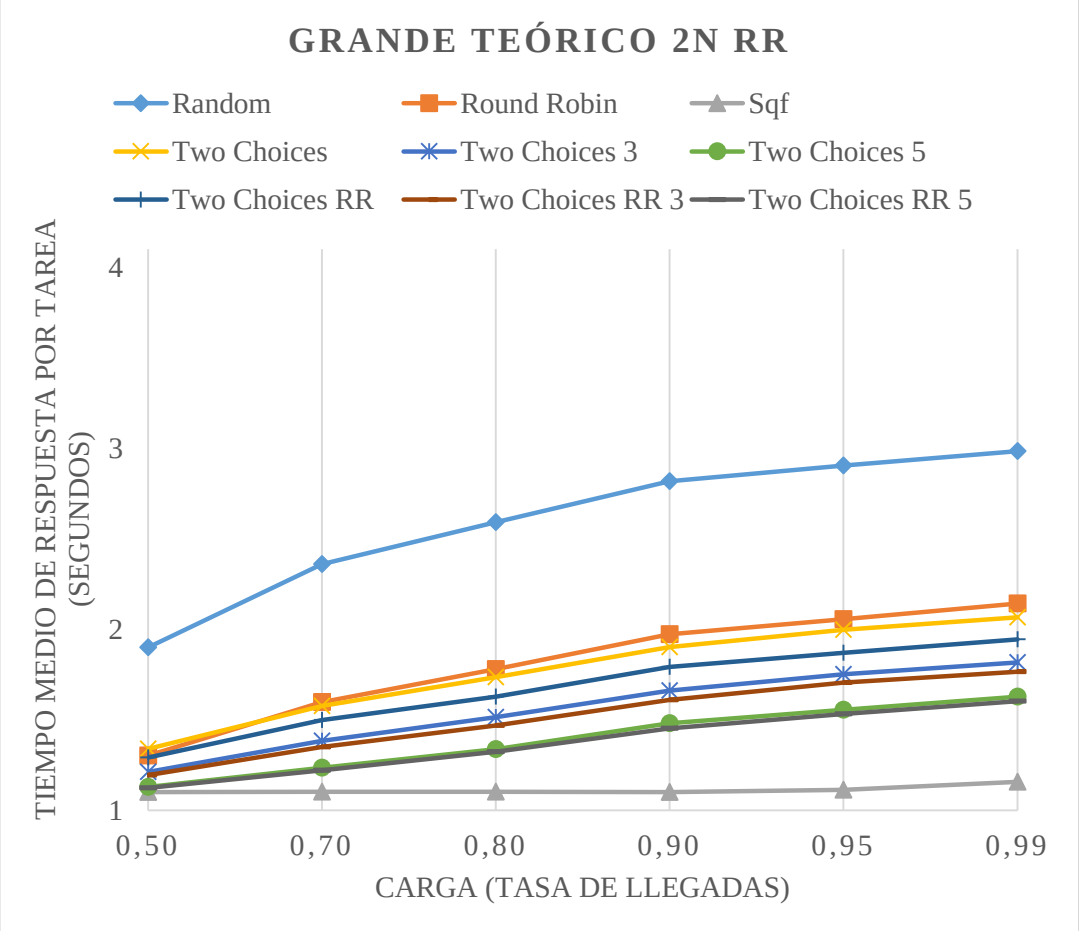


GRÁFICA 30: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA GRANDE 1N*

4.11. EXPERIMENTACIÓN CON LA PLATAFORMA GRANDE DE DOS NIVELES: EVALUACIÓN TEÓRICA Y REAL

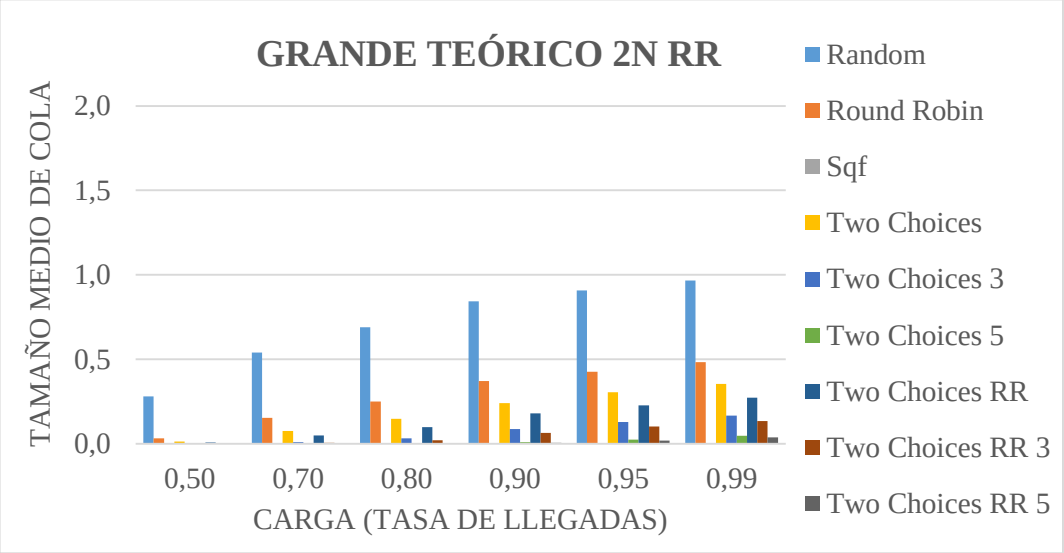
En este último punto de experimentación, se van a evaluar los algoritmos en la plataforma grande de dos niveles de los puntos 4.5 y 4.6. De la misma forma que en evaluaciones anteriores, las propiedades de las tareas como también el número que se generan de éstas coinciden con las del punto 4.7.

Una vez dicho esto, las gráficas 31 y 32 representan los valores obtenidos en la evaluación teórica de esta plataforma con el algoritmo *round robin* en el dispatcher de peticiones.



GRÁFICA 31: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA GRANDE DE 2N Y ROUND ROBIN*

Los tiempos medios de respuesta por tarea de la gráfica 31 son exactamente iguales a los obtenidos para la misma evaluación en la plataforma mediana de un nivel (gráfica 25). Por tanto, a raíz de los resultados, se puede afirmar que a nivel teórico dedicar un host del clúster para que se comporte de dispatcher de clúster no mejora el rendimiento. Además, cabe destacar que el tiempo medio de respuesta se ha reducido respecto de la plataforma mediana al disponer de más hosts para ejecutar tareas.

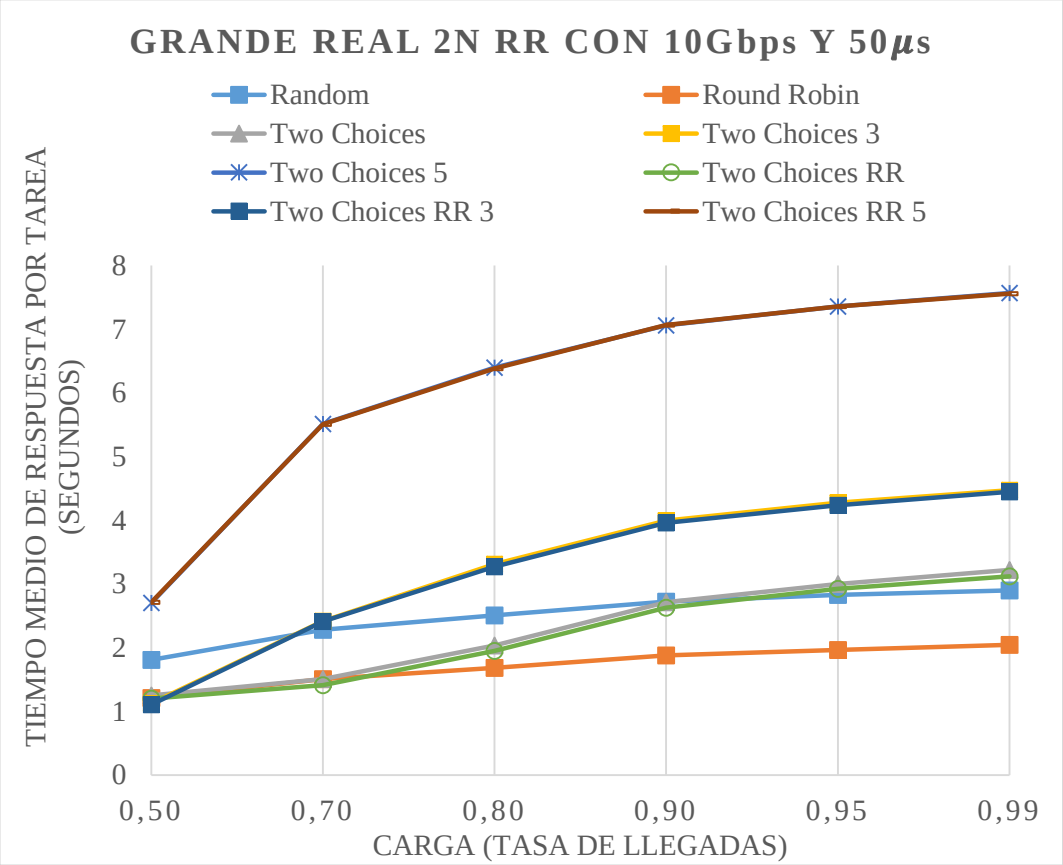


GRÁFICA 32: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA GRANDE 2N Y ROUND ROBIN*

Como se ve en la gráfica 32, el comportamiento de las colas de los host en la evaluación teórica ha sido la misma que para la plataforma grande de un nivel. Es por ello que para no duplicar información, se toma como buena la descripción dada para la gráfica 8.

Ahora que ya se ha descrito la evaluación teórica en la plataforma grande de dos niveles cuando el algoritmo del dispatcher de peticiones es *round robin*, se van a analizar los resultados obtenidos para la evaluación real. Los datos de la evaluación real con el modelo de links a 1Gbps no se han podido incluir ya que no se ha conseguido que la simulación finalice. Por tanto, únicamente se incluyen las gráficas 33 y 34 correspondientes a la evaluación con links a 10Gbps.

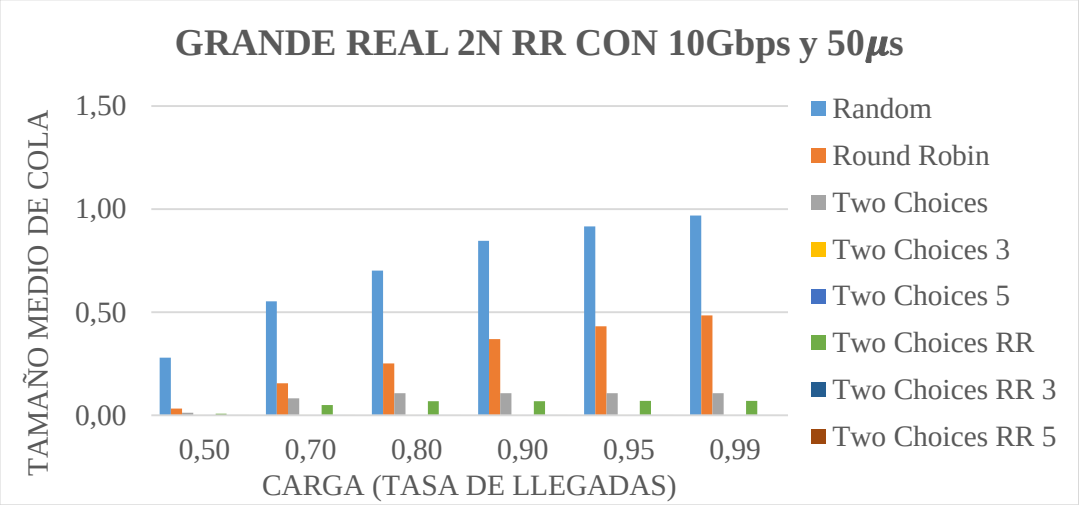
Lo primero a destacar de la gráfica 33 es que los tiempos de respuesta por tarea se reducen notablemente respecto de la misma evaluación en la plataforma grande de un nivel (gráfica 29). Por tanto, a diferencia de lo que establece la evaluación teórica en esta plataforma, la incorporación de un host encargado de distribuir las tareas dentro de los clústeres produce que el rendimiento mejore drásticamente en la evaluación real. Como se ha explicado para la misma evaluación en la plataforma mediana, esta mejora se debe a que ahora el dispatcher de peticiones distribuye las tareas a medida que le llegan de manera *round robin*, por lo que las tareas en la evaluación real ya no pierden tiempo esperando en la cola del dispatcher de peticiones a ser distribuidas.



GRÁFICA 33: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA GRANDE 2N Y ROUND ROBIN*

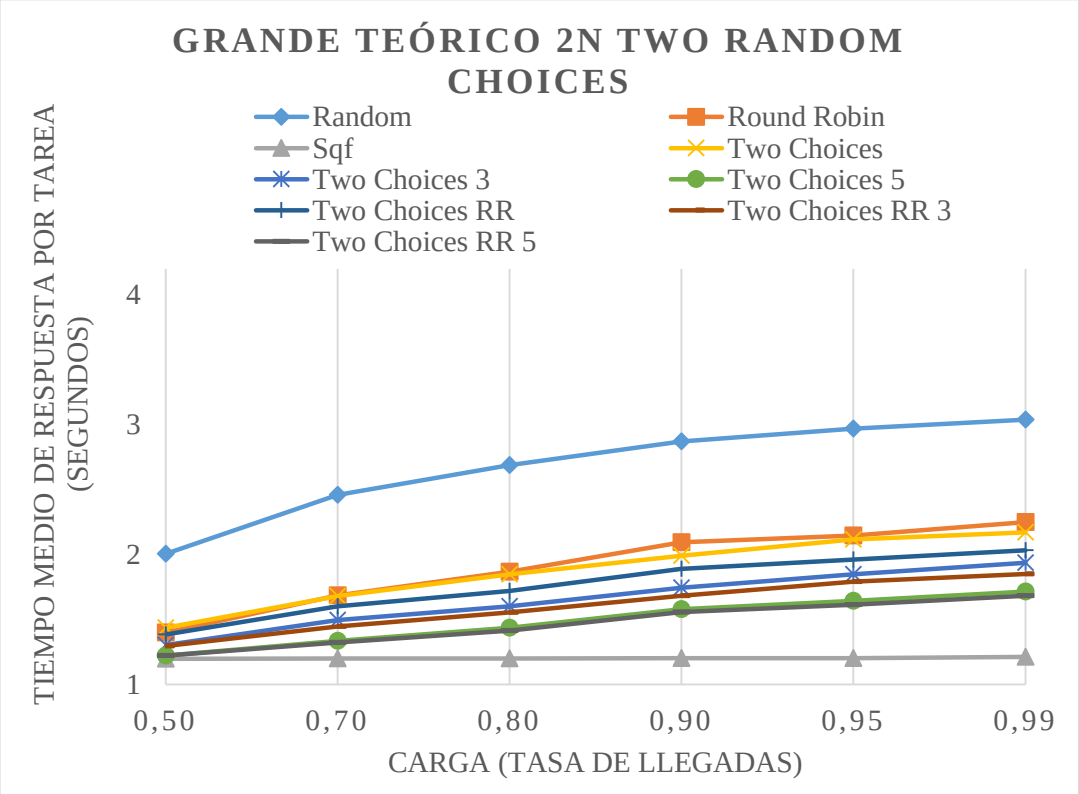
Otra faceta a destacar de la evaluación anterior es que el algoritmo *round robin* ya no es el mejor para todas las tasas de llegadas; para tasas de 0,5 el mejor es *two choices RR 3*, a partir de 0,5 y hasta 0,75 es el *two choices RR* y desde 0,75 hasta 0,99 *round robin*. Esto sucede por la mejora de la red, que permite que para tasas de llegada bajas, los algoritmos *two choices RR 3* y *two choices RR* puedan determinar el mejor de entre los host que comparan sin que ello suponga la espera de tareas en las colas de los dispatcher de clúster.

Finalmente, comentar que a partir de la tasa de llegadas de 0,75 el algoritmo *round robin* vuelve a ser con el que se obtienen mejores tiempos de respuesta medio. Al aumentar la tasa de llegadas, las tareas llegan más juntas en el tiempo, pero en cambio, los algoritmos *two choices RR y two choices RR 3* siguen requiriendo el mismo tiempo para enviar las tareas de petición por la red y recibir la respuesta (ya que la red sigue siendo la misma), por lo que se acumulan tareas en la cola del dispatcher de clúster y hay una bajada de rendimiento.



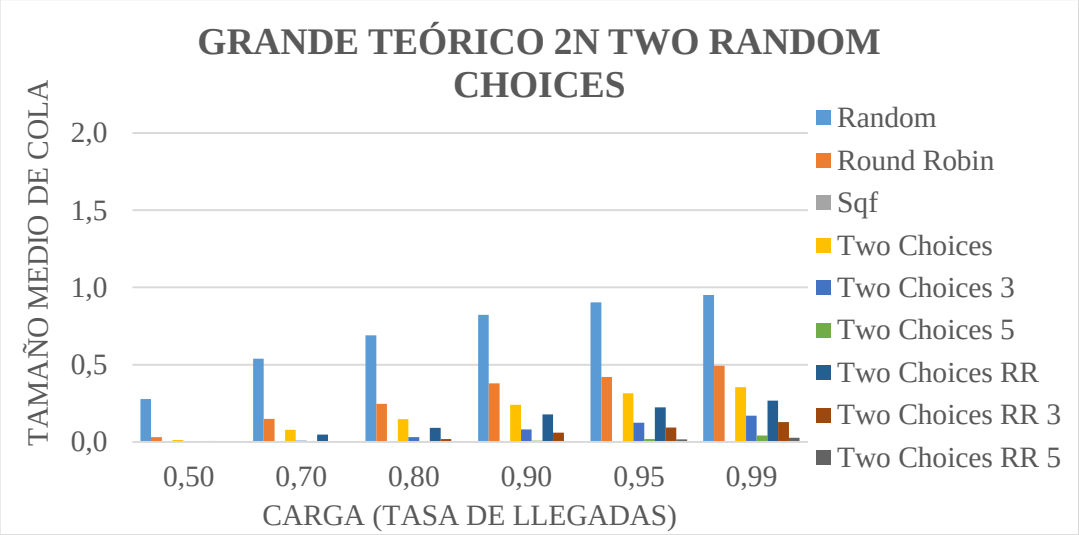
GRÁFICA 34: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA GRANDE 2N Y ROUND ROBIN*

Una vez descritas las evaluaciones teóricas y reales con la plataforma grande de dos niveles y *round robin* en el dispatcher de peticiones, se van a evaluar los resultados cuando el algoritmo encargado de distribuir las tareas entre los clústeres es el *two random choices*. Los resultados teóricos de ésta evaluación se muestran en las gráficas 35 y 36, mientras que los datos obtenidos en la evaluación real en las gráficas 37, 38, 39 y 40.



GRÁFICA 35: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA GRANDE DE 2N Y TWO RANDOM CHOICES*

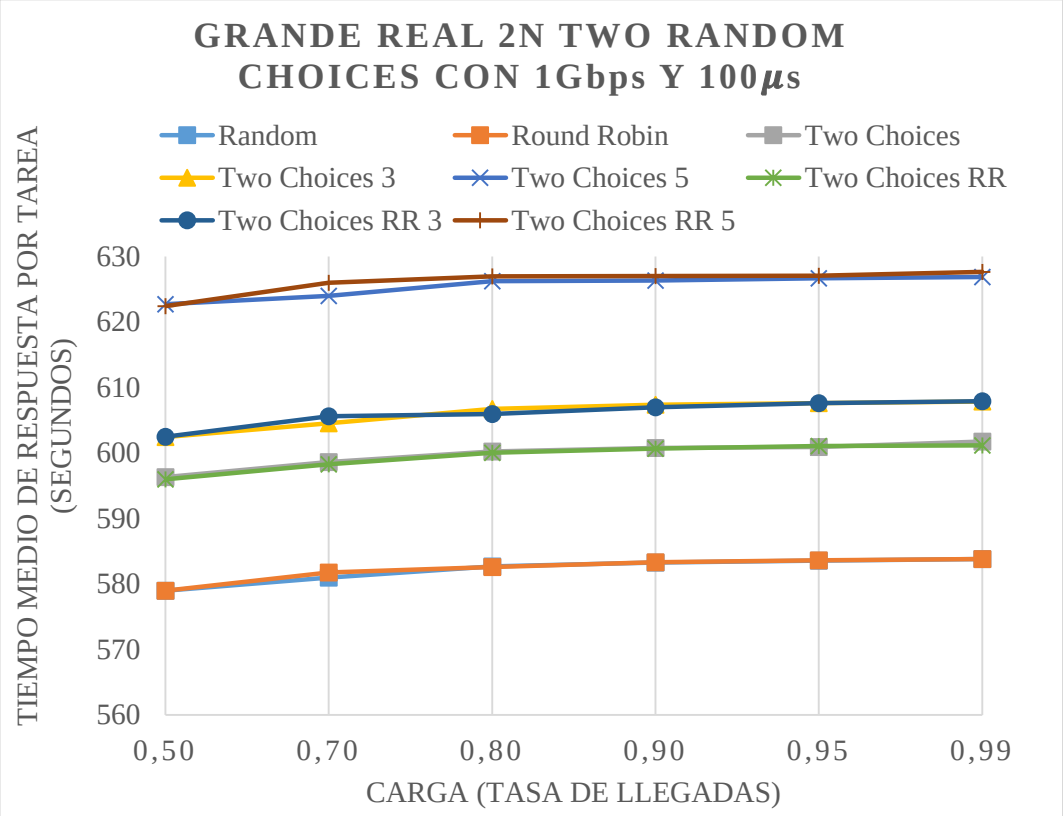
Como ya sucediera para las evaluaciones en la plataforma mediana, en la evaluación teórica con la plataforma grande de dos niveles se obtiene el mismo rendimiento independientemente de si se usa *two random choices* o *round robin* en el dispatcher de peticiones. Asimismo, al aumentar el número de host y no variar el número de tareas generadas para la simulación, el tiempo medio de ejecución por tarea se reduce respecto de la misma evaluación en la plataforma mediana.



GRÁFICA 36: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN TEÓRICA CON LA PLATAFORMA GRANDE 2N Y TWO RANDOM CHOICES*

Además, a partir de las gráficas 35 y 31 se puede constatar que para la evaluación teórica, situar en el dispatcher de peticiones uno u otro algoritmo no tiene consecuencia alguna en el rendimiento.

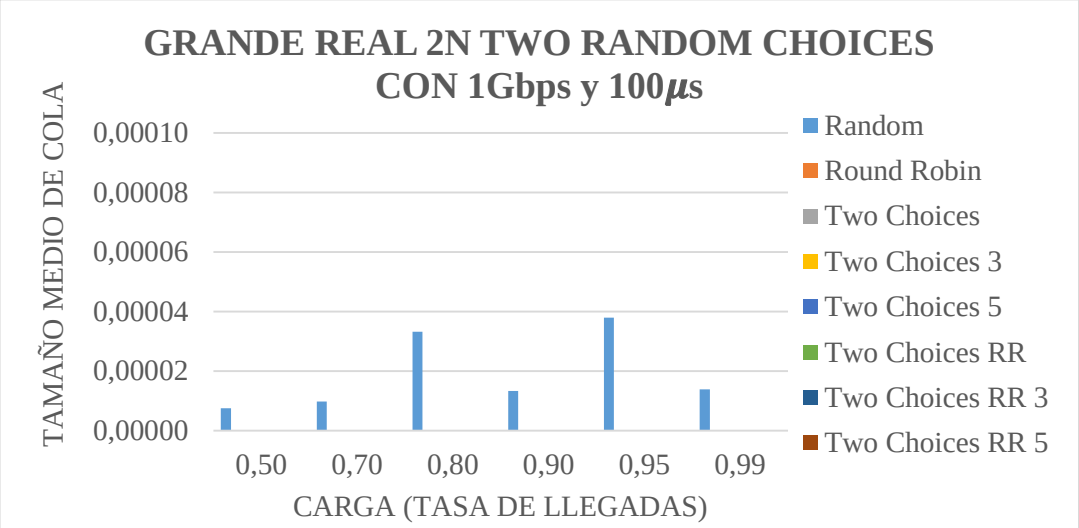
Una vez descritas las características más importantes de la evaluación teórica, mediante las gráficas 37, 38, 39 y 40 se va a describir el comportamiento de los algoritmos en la evaluación real con la plataforma grande de dos niveles y *two random choices* en el dispatcher de peticiones.



GRÁFICA 37: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA GRANDE 2N Y TWO RANDOM CHOICES*

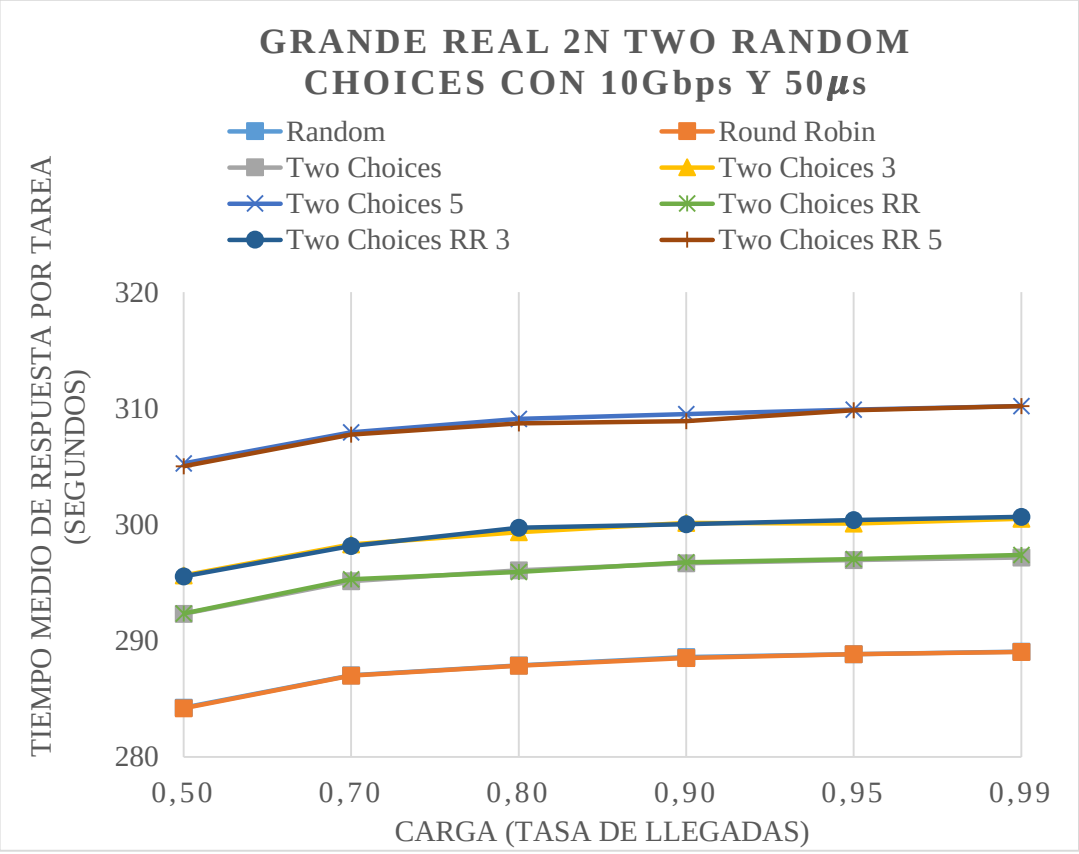
Lo primero que se puede ver es que una vez más los resultados extraídos no tienen nada que ver con los obtenidos en la evaluación teórica. Cuando en la plataforma grande de dos niveles se emplea el algoritmo *two random choices* en el dispatcher de peticiones, el rendimiento del sistema disminuye considerablemente en comparación con la utilización de *round robin*.

Asimismo, en la evaluación de la gráfica 37 se han obtenido mejores tiempos con los algoritmos del tipo *two random choices* que los obtenidos en la plataforma grande de un solo nivel (gráfica 27) con los mismos algoritmos. En cambio, con *random* y *round robin*, los resultados en esta plataforma han sido peores que los de la plataforma de un nivel. Esto se debe al algoritmo *two random choices* del dispatcher de peticiones, que por cada tarea que llega al sistema, consume tiempo en determinar el clúster de destino.

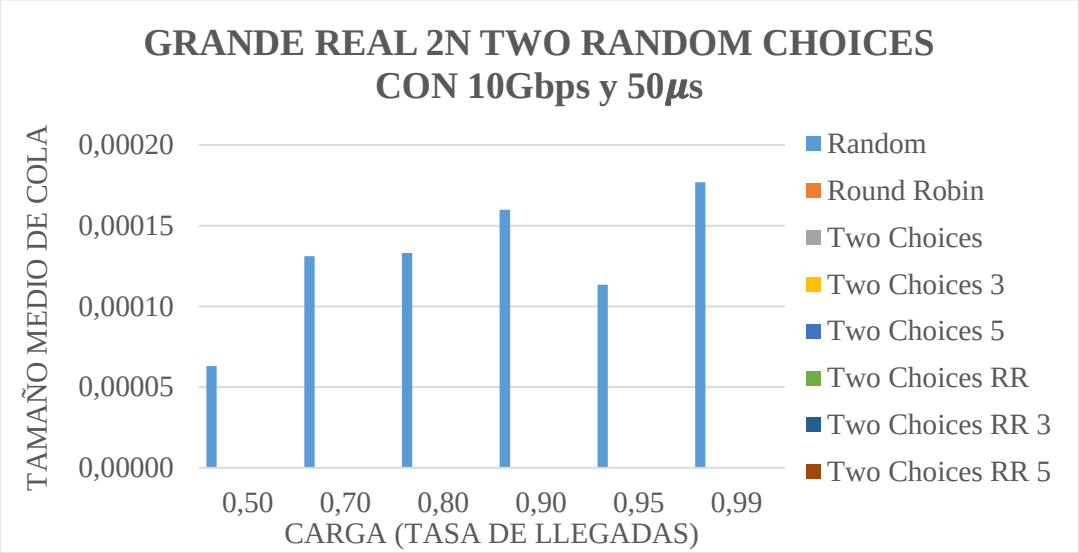


GRÁFICA 38: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 1Gbps CON LA PLATAFORMA GRANDE 2N Y TWO RANDOM CHOICES*

Finalmente, se puede ver que al igual que ha sucedido en todas las evaluaciones anteriores, al aumentar el ancho de banda, el tiempo medio de respuesta por tarea disminuye.



GRÁFICA 39: *TIEMPO MEDIO DE RESPUESTA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA GRANDE 2N Y TWO RANDOM CHOICES*



GRÁFICA 40: *TAMAÑO MEDIO DE COLA EN LA EVALUACIÓN REAL CON LINKS A 10Gbps CON LA PLATAFORMA GRANDE 2N Y TWO RANDOM CHOICES*

4.12. RESUMEN DE LOS RESULTADOS OBTENIDOS

En este punto se van a resumir todos los resultados que se han obtenido de la experimentación anterior.

Con el fin de facilitar la comprensión, se va a diferenciar entre las evaluaciones realizadas sobre la plataforma de un nivel y las de dos niveles.

Lo primero a destacar, es que para todas las evaluaciones, el sistema se ha enfrentado a una llegada de 100.000 tareas y además, se han realizado 10 ejecuciones por tasa de llegadas y algoritmo.

A partir de las evaluaciones realizadas, se llega a la conclusión de que los resultados obtenidos en la simulación teórica no tienen nada que ver con los obtenidos de la evaluación con la red real, independientemente de si la plataforma que se está utilizando es de uno o dos niveles.

En las plataformas de un nivel, cuando se realiza la simulación teórica, el algoritmo con el que el sistema obtiene peor rendimiento es el *random*, mientras que el *shortest queue first* obtiene los mejores resultados. Esto se debe a que en la evaluación teórica los mensajes de petición tienen un coste despreciable, lo que permite al algoritmo *shortest queue first* seleccionar el host más desahogado sin apenas penalización. Otro aspecto que llama la atención en las evaluaciones teóricas con la plataforma de un nivel, es que el algoritmo *round robin* es el segundo que peor rendimiento tiene. En un principio esto puede parecer extraño (debido a la política que sigue éste algoritmo para distribuir las tareas), pero el no tener en cuenta el estado de carga de los host es justamente lo que lo penaliza. Por todo esto, para la evaluación teórica de un nivel, cuanto mejor informado esté el algoritmo sobre el estado de las colas de los host, mejor va a ser el rendimiento que obtenga el sistema. Otro aspecto a destacar es que

a medida que la tasa de llegadas aumenta, el tiempo de respuesta medio también lo hace para todos los algoritmos.

Por último, para terminar de comentar las evaluaciones teóricas de un nivel, destacar que hay correspondencia entre los valores del tamaño medio de cola en los host y el tiempo medio de ejecución por tarea; los algoritmos que preguntan a los hosts sobre su cola consiguen distribuir mejor las tareas, al asignarlas a los hosts menos saturados, reduciendo así el tamaño de las colas y aumentando la cantidad de tareas ejecutadas por unidad de tiempo. Además, a medida que se añaden más host al sistema y el número de tareas de la simulación no se modifica, el tiempo medio de ejecución por tarea se reduce.

Respecto a las evaluaciones teóricas realizadas con las plataformas de dos niveles, se ha llegado a la conclusión de que dedicar un host dentro del clúster que haga de dispatcher no mejora el rendimiento. Igualmente, no hay diferencia entre usar *round robin* o *two random choices* en el dispatcher de peticiones, ya que el rendimiento que se consigue es el mismo. Como ya sucediera en la evaluación teórica con las plataformas de un nivel, al aumentar el número de host y no variar el número de tareas, el tiempo medio por tarea se reduce.

En la evaluación real con las plataformas de un nivel, los tiempos medios de servicio por tarea han sido mayores en la red de 1Gbps de ancho de banda y 100μs de latencia que en la de 10Gbps y 50μs. Además, a diferencia de lo obtenido en la evaluación teórica, el algoritmo *shortest queue first* es el que peores tiempos logra y, el *round robin*, con el que mejor resultados se obtiene para casi todas las plataformas y tasas de llegada. Esto se debe a que transferir las peticiones y respuestas a través de la red tiene un coste en la evaluación real, por lo que deja de ser eficiente preguntar a cada host del sistema cada vez que llega una tarea.

Asimismo, en las evaluaciones reales el rendimiento empeora al haber únicamente un dispatcher de peticiones, ya que con los algoritmos que preguntan a los host sobre el estado de sus colas, el dispatcher pasa la mayor parte del tiempo enviando y recibiendo mensajes y no distribuyendo tareas. Por tanto, para todas las evaluaciones reales en la plataforma de un nivel (menos en la de la plataforma pequeña con links a 10Gbps), a medida que el algoritmo realiza más solicitudes sobre el estado de las colas en los host, mayor es el tiempo medio de ejecución de una tarea en el sistema.

Finalmente, cabe destacar que tampoco hay diferencia entre el rendimiento obtenido por los algoritmos *two random choices* y *two random choucies-round robin* y que además, en la evaluación real de un nivel, a medida que la plataforma en la que se evalúan los algoritmos es más grande, el rendimiento que se obtiene es peor. Esto se debe a la tasa de llegadas, que al depender de los hosts que tiene el sistema, hace que las tareas llegue más juntas en el tiempo, por lo que la red se congestiona y se reduce considerablemente el rendimiento.

Los datos obtenidos en la evaluación real con las plataformas de dos niveles tampoco tienen nada que ver con los de las evaluaciones teóricas. Aun así, los tiempos de respuesta por tarea en estas plataformas se han reducido respecto a los obtenidos en la plataforma de un nivel con la misma evaluación. Por tanto, a diferencia de lo que se deducía en la evaluación teórica, la incorporación de un host encargado de distribuir las tareas dentro de los clústeres hace que el rendimiento mejore.

Para finalizar, destacar la importancia de que el dispatcher de peticiones en estas arquitecturas no utilice algoritmos que determinen el clúster de destino para una tarea a partir del estado de las colas de los host; por el contrario, es importante que las tareas se distribuyan rápidamente desde el dispatcher de peticiones, por ejemplo, mediante el uso del algoritmo *round robin*.

CAPÍTULO 5: PLANIFICACIÓN Y PRESUPUESTO

En este penúltimo capítulo del estudio se va a exponer de manera detallada la planificación que se ha seguido y el presupuesto necesario para la realización del Trabajo de Fin de Grado.

5.1. PLANIFICACIÓN

En este apartado se va a exponer la planificación seguida para realizar las diferentes tareas de las que se ha compuesto este Trabajo de Fin de Grado.

Para facilitar la planificación de las tareas realizadas, se han agrupado en fases de la siguiente manera:

- Recopilación y búsqueda de información: entre este punto y el siguiente se engloban todas las tareas realizadas antes de la experimentación. Aquí se incluye el estudio de los sistemas distribuidos, la búsqueda de una herramienta que permitiese realizar las pruebas deseadas, búsqueda de algoritmos de distribución de carga, etc.

- Desarrollo e implementación: dentro de este conjunto se tiene en cuenta el desarrollo del código, su puesta en funcionamiento y las diferentes pruebas y correcciones realizadas con el objetivo de asegurar que el comportamiento del sistema era el deseado.

- Experimentación: aquí se incluyen todas las tareas realizadas para llevar a cabo los diferentes experimentos de los que se ha compuesto el estudio. Estas son la obtención de los datos de la simulación, su tratamiento y evaluación.

- Documentación: esta es la última fase, en la que se incluye la redacción de este documento. La información de esta memoria se ha ido recopilando durante todo el proyecto, pero la redacción final se ha hecho en esta fase.

En la tabla 3 se muestra la distribución en el tiempo de las fases descritas anteriormente. Para cada una de ellas, se define el ciclo en el que se ha realizado, una fecha de inicio, una fecha de fin y los días invertidos en su realización. Se debe de tener en cuenta que no ha sido posible dedicarle las mismas horas cada día al proyecto, pero por norma general, se han dedicado una media de 2 horas al día.

FASE DEL PROYECTO	CICLO	FECHA INI	FECHA FIN	DÍAS
Recopilación y búsqueda de información	1	21/09/2015	18/10/2015	27
Desarrollo e implementación: desarrollo	2	19/10/2015	27/03/2016	143*
Desarrollo e implementación: puesta en funcionamiento, pruebas y corrección de errores	3	28/03/2016	14/04/2016	17
Experimentación: obtención de datos	4	15/04/2016	24/04/2016	9
Experimentación: tratamiento y evaluación de los datos	5	25/04/2016	02/05/2016	7
Documentación	6	20/05/2016	17/06/2016	28

TABLA 3: *PLANIFICACIÓN DEL PROYECTO*

* En las vacaciones de Navidad, el ciclo 2 estuvo parado durante 17 días. Además, entre el ciclo 5 y 6 también se dejaron 18 días libres.

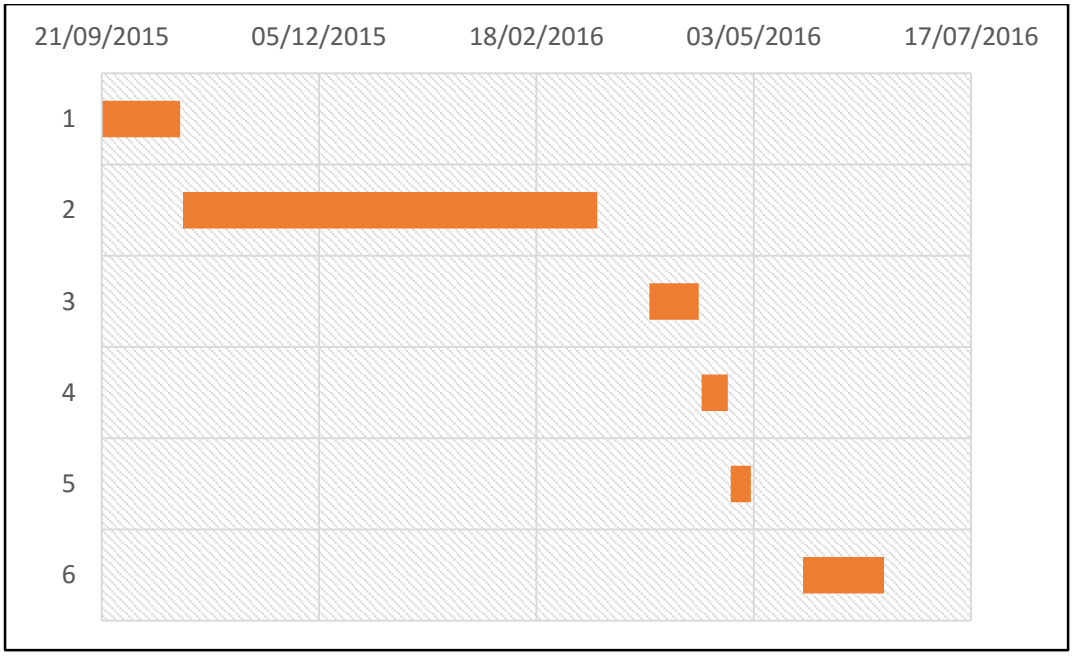


ILUSTRACIÓN 12: *DIAGRAMA DE GANTT DE LA PLANIFICACIÓN DEL PROYECTO*

La planificación recién descrita refleja cómo el proyecto se ha ido desarrollando. En un primer momento, no estaba previsto que se alargase tanto en el tiempo, pero debido a que era la primera vez que se utilizaba SimGrid y a la complejidad del sistema que se quería reproducir, el ciclo 2 se ha alargado más de lo que se esperaba.

En total se han dedicado 231 días al Trabajo de Fin de Grado. Teniendo en cuenta que se ha trabajado una media de 2 horas por día, el tiempo dedicado a este trabajo ha sido de 462 horas.

5.2. PRESUPUESTO

En este apartado se va a detallar el presupuesto necesario para llevar a cabo un estudio de estas características. Para una mejor comprensión, se han diferenciado tres tipos de gastos: los de hardware, los de software y los de personal.

5.2.1. GASTOS DE HARDWARE

Para la realización del proyecto, ha sido utilizado un portátil Apple MacBook Pro del año 2012 con las siguientes características técnicas:

- Procesador: Intel Core i5-3210M 2,5GHz.
- Memoria RAM: 8GB 1600 MHz DDR3.
- Disco Duro: 128GB SSD.

Un equipo de estas características está valorado actualmente en 700€ con IVA. En la siguiente tabla se muestra el coste detallado del hardware utilizado.

HARDWARE	PRECIO + IVA	UNIDADES	COSTE
Portátil Apple	700 €	1	700 €
TOTAL			**700 €**

TABLA 4: *GASTOS DE HARDWARE DEL PROYECTO*

5.2.2. GASTOS DE SOFTWARE

Respecto a la parte de software, estos son los programas que se han utilizado para la realización del proyecto:

- Sistema Operativo: Linux Ubuntu 14.04.
- Software: yEd, Microsoft Office 365 y SimGrid.

En la tabla 5, se refleja el coste de cada uno de los productos de manera individual y la suma de todos ellos como coste total del software.

SOFTWARE	PRECIO + IVA	UNIDADES	COSTE
Linux Ubuntu 14.04	0 €	1	0 €
Microsoft Office 365 *	0 €	1	0 €
SimGrid 3.11	0 €	1	0 €
yEd	0 €	1	0 €
TOTAL			**0 €**

TABLA 5: *GASTOS DE SOFTWARE DEL PROYECTO*

* El coste real de Microsoft Office 365 para cualquier persona que quiera obtenerlo no son 0 €. En cambio, su coste para el proyecto ha sido de 0 € ya que

la universidad a través del programa MSDNAA licencia a estudiantes programas de pago a coste 0 €.

5.2.3. GASTOS DE PERSONAL

El grueso del presupuesto lo forma el gasto de personal, que ha sido calculado a partir de la planificación descrita en el apartado 5.1. Como se describe en ese apartado, el número de horas dedicadas al proyecto han sido un total de 462. Por tanto, fijando la hora de trabajo en 22€ brutos, se ha obtenido que el gasto en personal ha sido de 10164€.

5.2.4. GASTOS TOTALES

Para finalizar, se detallan todos los gastos anteriores y se muestra el presupuesto total necesario para llevar a cabo este estudio.

TIPO DE GASTO	COSTE
Gastos en Hardware	700 €
Gastos en Software	0 €
Gastos en Personal	10164 €
TOTAL	10864 €

TABLA 6: *GASTOS TOTALES DEL PROYECTO*

CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se exponen las conclusiones a las que se ha llegado a partir de los datos obtenidos en la experimentación, cumpliendo así con los objetivos marcados al inicio de este trabajo. También, se van a proponer ideas para posibles trabajos futuros que tomen éste como punto de partida, o mejoras que se podrían realizar sobre lo que se ha hecho.

6.1. CONCLUSIONES

En este Trabajo de Fin de Grado, se ha llevado a cabo un estudio para determinar el comportamiento de diferentes algoritmos de distribución de la carga bajo una serie de circunstancias. A continuación, se van a exponer las conclusiones a las que se ha llegado a partir del estudio realizado. De esta forma:

- No es la velocidad, es la latencia. Se ha observado que, a partir de un ancho de banda, las mejoras en el rendimiento son imperceptibles y la clave de todo pasa a estar en la latencia. La latencia es el tiempo que tardan los mensajes en llegar desde un punto a otro de la red.

- Correspondencia entre la cola que se genera en los host y el rendimiento del sistema con la red teórica. Cuando el envío de información a través de la red tiene un coste despreciable, los algoritmos con los que se obtiene mejor rendimiento son los que tienen en cuenta el tamaño de las colas de los hosts. Por tanto, es imprescindible encontrar el equilibrio entre la arquitectura de red instalada y el algoritmo escogido para beneficiarse de la mejora que aporta la utilización de algoritmos que se informan sobre el estado del sistema antes de tomar una decisión.

- La velocidad a la que llegan las tareas al sistema influye considerablemente en su rendimiento, especialmente con la plataforma de un nivel. En las gráficas obtenidas, se observa que a medida que las tareas llegan más juntas, mayor es el tiempo que el sistema tarda en despacharlas. Una posible mejora a este problema, es crear más de un dispatcher de peticiones y que, además, el algoritmo encargado de distribuir las tareas varíe en función de la carga que tenga el sistema.

- Disponer únicamente de un dispatcher de peticiones para distribuir la carga entre los clústeres no es una buena solución. Cuando únicamente se dispone de un dispatcher de peticiones, los algoritmos del tipo *two random choices* no dan buenos resultados, ya que no permiten distribuir otras tareas mientras se encuentran preguntando a los host sobre el tamaño de sus colas. Por otro lado, es muy importante encontrar el número óptimo de hosts a los que preguntar sobre el tamaño de su cola para que el rendimiento sea superior al que se obtiene con la utilización de los algoritmos *random* y *round robin*.

- A raíz de los resultados obtenidos, se puede afirmar que no hay diferencia significativa de rendimiento entre usar el algoritmo *two random choices* o *two random choices-round robin*.

- En las plataformas de dos niveles, el dispatcher de peticiones no debe usar algoritmos que determinen el clúster de destino de una tarea a partir del estado de las colas. Es importante que el algoritmo que se utilice distribuya rápidamente las tareas a los clústeres, por ejemplo, con el uso del algoritmo *round robin* y que sea ya allí (en los clústeres) donde se planifique la asignación.

- Si se desea aumentar el rendimiento de un sistema de un nivel, simplemente basta con aumentar el número de host, siempre y cuando la velocidad a la que llegan las tareas, la cantidad y las características de éstas no varíen respecto del sistema que se está actualizando. Otra forma de aumentar el rendimiento es convertir el sistema en uno de dos niveles.

- De forma teórica, cuanta más información tenga el algoritmo sobre el estado del sistema para tomar una decisión, mejor va a ser el rendimiento. En la realidad esto no sucede y por ejemplo, el algoritmo *shortest queue first* es muy ineficiente. Es muy importante estudiar bien la carga media a la que se enfrenta el sistema para determinar así si a través del uso de estos algoritmos el rendimiento del sistema mejora o empeora.

Por todo lo expuesto en este trabajo, se puede afirmar que el proyecto ha cumplido con los objetivos establecidos.

6.2. TRABAJOS FUTUROS

En este apartado, se va a hablar de posibles trabajos que se pueden desarrollar tomando éste como base.

La primera propuesta consiste en realizar el mismo estudio, analizando el comportamiento de los algoritmos cuando la latencia de la red va variando. Sería muy interesante ver cómo influye en este caso la latencia en el rendimiento del sistema y encontrar los límites en los que el aumento del ancho de banda produce mejoras imperceptibles debido a que la clave del proceso pasa a estar en la latencia.

Otra posible propuesta es la evaluación de los algoritmos cuando el sistema dispone de más de un dispatcher de peticiones y además, el algoritmo de distribución de la carga en los dispatcher varía dependiendo del ritmo de llegada de las tareas al sistema.

También sería interesante estudiar cómo influye en el rendimiento la utilización de una red troncal dentro del clúster a la que todos los host se conectan, en vez de lo realizado en este estudio, donde se ha definido un cable individual desde cada host al router del clúster.

Igualmente se plantea efectuar la evaluación con diferente número de tareas. En este estudio, el generador de peticiones ha creado en todas las evaluaciones 100.000 tareas, por lo que evaluar las plataformas creadas y los algoritmos implementados con otra cantidad de tareas podría ayudar a completarlo.

Por último, pero no por ello menos importante, se plantea la creación de simulaciones con plataformas más grandes y con otro tipo de algoritmos.

ANEXO I: COMPETENCIA EN INGLÉS

I.I. INTRODUCTION

Distributed systems appeared few time ago in the brief history of informatics. Computers are continuously getting cheaper and smaller. This makes the same room able to host more computers at a cheaper price every day. Nowadays thousands of computers can be placed where before only one it could be. Their price and size have also reduced exponentially and what it is more important, their speed and efficiency have increased in a vertiginous manner. Communication through the network consumes clock cycles and for this reason a slow computer devoted most of its time to that task instead to the execution of the user program. Therefore, not many years ago, with the costs and performances of old CPUs, communicating through the network was not an option. However, thanks to the advances achieved in connecting networks, connecting computers has become something easy and cheap.

Tanenbaum defines a distributed system as "a set of independent computers which presents itself to the user like a unique and consistent system". In this definition two essential points are established: firstly, the use of the word "independents", which means that from the point of view of their architectures, the computers are able to operate independently; secondly, related to "a unique system", which means that software allows this set of interconnected computers to show themselves like only one to the users of the system. This is known as SSI (Single System Image) and, as we will see later on in this study, its main task is to design distributed systems which are easy to operate and maintain.

But…Why to make use of distributed systems? The use of distributes systems gives us the following advantages:

- Sharing of resources; a distributed system allows sharing resources both of hardware and software.
- Openness; they use to be designed on standard protocols that allow combining both equipment and software from different sellers.
- Concurrency; in a distributed system various processes can operate at the same time across different computers of the network.
- Scalability; the capacity of the system can be incremented by adding new resources to cover additional demands on the system.
- Tolerance to failures; most distributed systems are able to provide services even in case of failures. The full loss of service occurs only in case of failure of the network.

But distributed systems are not exempt of problems and amongst the main disadvantages of using a distributed approach we can find the following:

- Complexity; to design, implement and utilise distributed software may be difficult. Distributed systems are much more complex than centralized ones.

- The network may lose messages and/or overloading; to cable the net again may be difficult and expensive.
- Security; facility of access from any point brings problems of security. Since traffic on the net may be subjected to undesirable situations it is more difficult to ensure data integrity.
- Handle; it is required more effort to manage and keep the system working properly, since there may be different types or versions of operational systems; the defects of a machine may get extended to others with unexpected consequences.

Within the distributed systems one of the most outstanding architectures is the one of the clusters. The main problem that they present, which affects considerably to their performance, is the distribution of load which is mostly trusted to a series of algorithms. Through these algorithms, distributed systems get the highest possible level of efficiency. Distribution of load consists of selecting the node or nodes where a determined task is executed, so that depending on this election, the time needed for executing the task is or is not minimized. This is the main aim of this study, the study of diverse algorithms for distributing the load.

To this purpose we have defined a general problem (figure 13) which consists of a custom entity that generates tasks and send them through the network to the entity *dispatcher*. The *dispatcher* receives the tasks of the customer and depending on what determines its implemented algorithm of distribution of load, the task is sent to one cluster or to another.

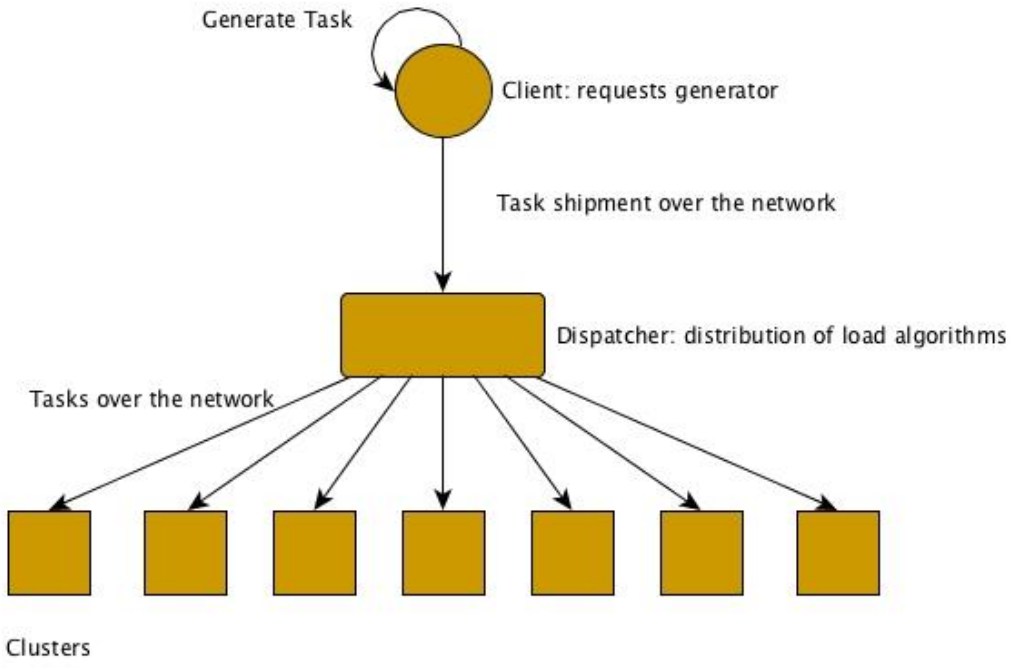


ILUSTRACIÓN 13: *GENERAL PROBLEM*

I.II. OBJECTIVES OF THE WORK

Given the problem of distribution of load in a cluster, the principal objective of this work is to study and determine the best algorithms for the distribution of load for a series of proposed configurations of clusters. Therefore, in general terms, the objectives in a more precise way are the following:

- **To study the distributed systems, focusing our attention on clusters**, their characteristics and the components they are composed of.

- **To study different algorithms that may be utilised for distributing the load in distributed systems**. To this purpose we will study a series of general concepts which will allow us to distinguish between diverse algorithms.

- **To study the basic concepts for the simulation on distributed systems, as a tool to guide in their design.**

- **To carry out a study of the framework of the tool for simulation SimGrid**, making evident its main features. We shall examine its structure by layers and specify the elements that are necessary to carry out a simulation.

- **To define different files of cluster platform in SimGrid.** We shall define several configurations of different clusters which will be implemented in the files of the platform SimGrid.

- **To implement the studied algorithms in SimGrid and carry out the simulation with each one of the defined platforms**. The algorithms chosen will be implemented in SimGrid and will be simulated with each one of the platform files created. Each simulation will be executed for different values of the rate of arrival or, what is the same thing, the number of tasks per second.

- **To obtain and compare the results in order to determine which algorithms are the best and under what circumstances**. The data obtained will be displayed on a series of graphics allowing the study of the reaction of the algorithms in respect to the mean time of execution of a task and to the number of tasks waiting to be executed.

I.III. RESULTS

In this Chapter we lay out the summary of the outcomes obtained from the previous experiment. With the aim of facilitating its understanding, we differentiate between the evaluations carried out on the one-level from the two-level platforms.

It has to be underlined that for all the evaluations, the system has faced with the arrival of 100.000 tasks and in addition, 10 executions have been done per each arrival rate and algorithm.

Based on the evaluations we reach the conclusion that the results obtained in the theoretic simulation are completely different from those obtained in the evaluation with the actual network, regardless the platform utilised is one-level or two-level.

When simulating theoretically with the one-level platforms, *random* is the algorithm which provides the worst performance of the system, while the *shortest queue first* is the one that gets the best results. This is due to the fact that in the theoretic evaluation the messages of request have an unappreciable cost, allowing the algorithm *shortest queue first* to select the most unloaded host with hardly any penalisation.

Another aspect calling our attention when evaluating the one-level platforms is that the algorithm *round robin* presents the second worst performance. In principle this might seem odd (due to the policy this algorithm follows for distributing the tasks), but what penalizes it is precisely not taking into account the state of load of the hosts. For all this, in the evaluation of the one-level platform, the better informed on the state of the queues of the hosts the algorithm is the higher the performance of the system will be. Another aspect to be underlined is that when the rate of arrivals increases so it does the mean time of response for all algorithms.

To conclude the comment on the theoretic evaluations of one-level, we should highlight that there is a correspondence between the values of the mean size of the queue in the host and the mean time of execution per task; the algorithms that ask hosts on their queue achieve a better distribution of tasks as they assign them to the less saturated hosts, reducing in this way the size of the queues while increasing at the same time the number of tasks executed per second. Besides, the more hosts are added to the system keeping the same number of tasks of the simulation, the lower the mean time of execution per task is.

In respect to the theoretic evaluations carried out with the two-level platforms, we have reached the conclusion that when dedicating a host to act as a dispatcher within the cluster, the performance does not increase. At the same time, there is no any difference between using *round robin* or *two random choices* in the dispatcher of requests, as the performance achieved is the same in both cases. As it happened in the evaluation of the one-level platforms when the number of host increases, keeping constant the number of tasks, the mean time per task reduces.

Regarding to the real evaluation with the one-level platforms, the mean times of service per task have resulted to be higher in the network of 1Gbps of bandwidth and 100 microseconds of latency than in the one of 10Gbps and 50 microseconds. Besides, otherwise to what was obtained in the theoretic evaluation, the algorithm *shortest queue first* is the one that gets the worst time while the *round robin*, on the contrary, gets the best results for most of the platforms and rates of arrival. This is due to the fact that transferring requests and answers through the network has a cost in the real evaluation and for this reason it is not efficient anymore to ask every host in the system at each time a task arrives.

Likewise, in the real evaluations the performance decreases as there is only one dispatcher of requests, since with the algorithms that ask to the hosts on the state of their queues, the dispatcher invests most of the time in sending and receiving messages instead of in distributing tasks. Therefore, for all the real evaluations on the one-level platform (except in the small platform with links at 10 Gbps), the more requests on the state of queues of hosts the algorithm does the higher the mean time of execution of a task in the system gets.

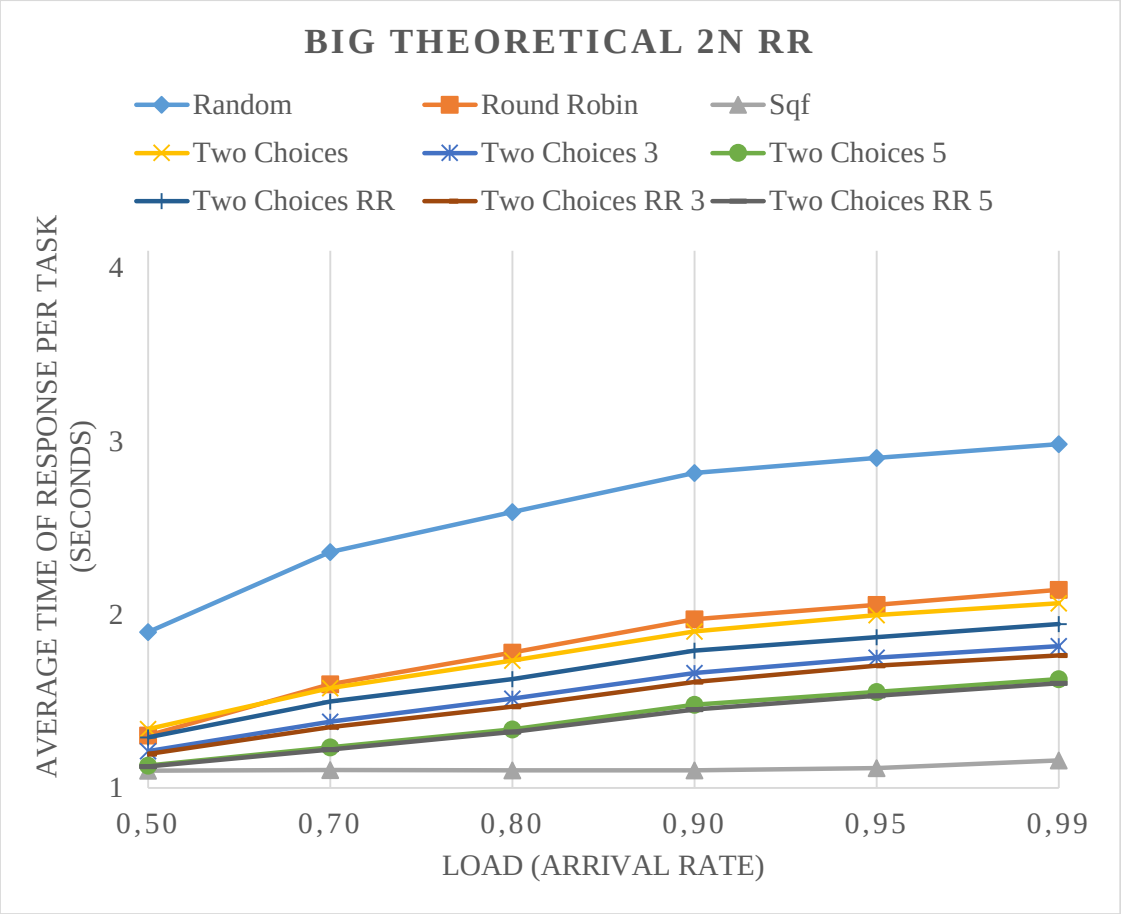
Finally, we should also underline that there is no any difference between the performances of the algorithms *two random choices* and *two random choices-round robin* and that, in addition, in the one-level real evaluation, the larger the platform on which the algorithms are evaluated is the worse its performance gets. This is due to the rate of arrivals that as it depends on the hosts the system has, it makes the tasks to arrive closer in time, driving the network to become congested and consequently the performance considerably reduced.

The data obtained in the real evaluation of the two-level do not match with those obtained in the theoretic evaluations either. Even so, in these platforms the times of answer for tasks have been reduced in respect to the ones obtained in the one-level platform for the same evaluation. Therefore, the incorporation of a host in charge of distributing the tasks within the clusters makes the performance to increase as opposed to what was deducted from the theoretic evaluation.

To conclude, to highlight the importance that in these architectures the dispatcher of requests should not utilise algorithms determining the cluster of destination of tasks based on the state of the queues of the host: on the contrary, it is important to make the tasks to be distributed from the dispatcher of requests, for example, by means of using the algorithm *round robin*.

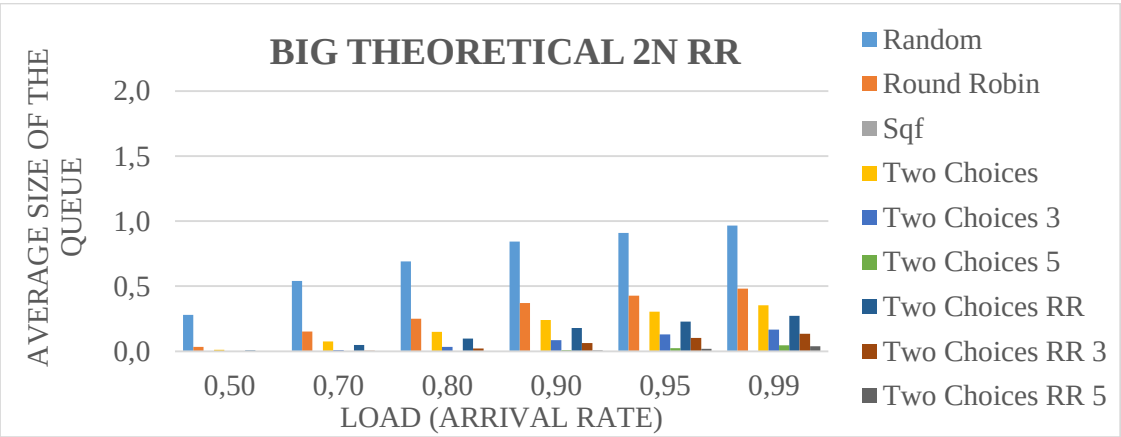
Graphics 41, 42, 43 and 44 show two examples of a simulation using the big theoretical two level platform and the real one with 10 Gbps connections. For both simulations, the algorithm being used in the request dispatcher is *round robin*.

As we said before, there is no any difference between using *round robin* or *two random choices* in the dispatcher of requests, as the performance shown on graphic 41 shows very little difference between those two algorithms.



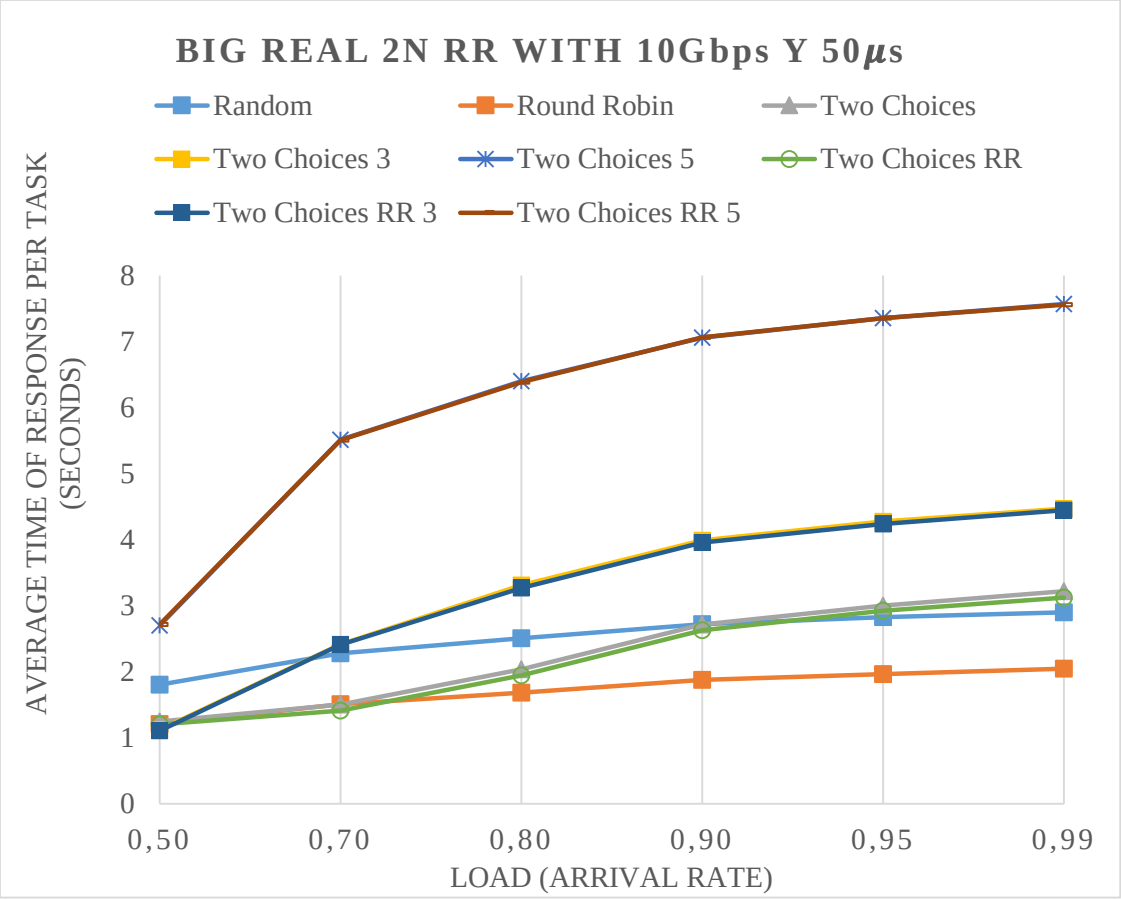
GRÁFICA 41: *AVERAGE TIME OF RESPONSE ON THE THEORETICAL EVALUATION WITH THE BIG TWO LEVEL PLATFORM AND ROUND ROBIN*

Again, the *shortest queue first* algorithm obtains the best results in the theoretical evaluation.



GRÁFICA 42: *AVERAGE SIZE OF THE QUEUE ON THE THEORETICAL EVALUATION WITH THE BIG TWO LEVEL PLATFORM AND RR*

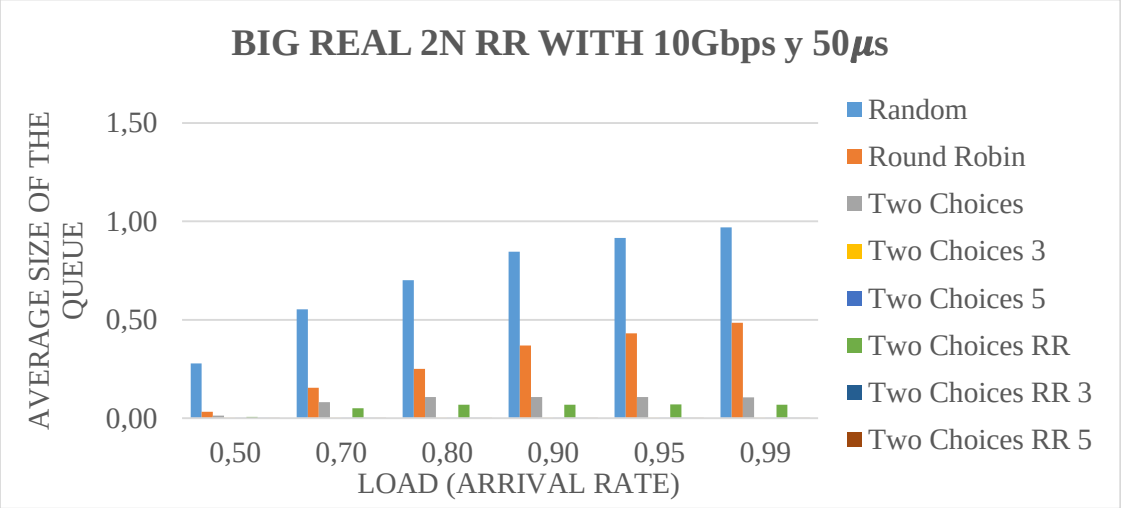
Another aspect calling our attention is that in the evaluation shown on graphic 42 the average size of the queues follows a pattern; the algorithms that ask hosts on their queue achieve a better distribution of tasks as they assign them to the less saturated hosts, reducing in this way the size of the queues while increasing at the same time the number of tasks executed per second.



GRÁFICA 43: *AVERAGE TIME OF RESPONSE ON THE REAL EVALUATION WITH 10Gbps LINKS AND THE BIG TWO LEVEL PLATFORM AND ROUND ROBIN*

Regarding the evaluation in graphic 43, the *round robin* algorithm is not the best for all arrival rates anymore; for the arrival rate of 0,5 the best algorithm is *two choices RR 3*, for arrival rates between 0,5 and 0,75 is the *two choices RR* and finally, from 0,75 through 0,99 is *round robin*. The network improvement allows the algorithms *two choices RR 3* and *two choices RR* to choose the best among all the hosts they compare without causing the queues at the cluster dispatcher to grow.

To end up with this chapter, it is important to emphasise that on the evaluation in graphic 44 the algorithms that ask hosts on their queue before taking any decision are those which make smaller queue sizes, on the other hand, algorithms like *random* or *round robin* are the ones that generate bigger queue sizes.



GRÁFICA 44: *AVERAGE SIZE OF THE QUEUE ON THE REAL EVALUATION WITH 10Gbps LINKS AND THE BIG TWO LEVEL PLATFORM AND ROUND ROBIN*

I.IV. CONCLUSIONES AND FUTURE WORKS

Over this chapter we lay out the conclusions we have reached from the data obtained in the experimentation, accomplishing in this way the objectives which were set up at the beginning of this work. In addition, we are going to propose some ideas to inspire some future works which could be undertaken taking this one as initial point of departure or some improvements that could be realised over what has already been described before.

I.IV.I. CONCLUSIONS

This End of Degree Work contains the study carried out to determine the behaviour of different algorithms for the distribution of load under a series of circumstances. The inference achieved through this study is as follows:

- It is not the speed, it is the latency. We have observed that from a determined bandwidth, the improvements in the efficiency are unperceivable and latency is becoming the key. Latency is the time that messages need to travel between two points over the network.

- Correspondence between the queue generated in the host and the efficiency of the system with the theoretical network. When sending information through the network has an unappreciable cost, the algorithms which get higher efficiency are those which take into account the size of the queues of the hosts. To benefiting from the improvements provided by algorithms that take into account the system state before making a decision it is therefore essential to find the balance between the installed architecture and the chosen algorithm.

- The rates at which tasks reach the system influence considerably its overall performance, especially in the one-level platforms. By observing the depicted graphics, we observe that the higher is the rate of tasks reaching the system the longer the system takes to dispatch them. A possible measure to smooth out this problem would consist of increasing the number of dispatchers and besides, making the algorithm in charge of the load distribution to vary depending on the load of the system.

- Having at our disposal only one dispatcher of requests to distribute the load amongst the clusters is not a good solution. When we have only one dispatcher of requests at our disposal, the *two random choices* type algorithms do not provide good results as they don't allow to distribute other tasks while they are in the process of interrogating the hosts on the size of their queues. On the other side, it is very important to find the balance on the number of hosts which to ask about their queue in order to get a higher performance than the one obtained by using the algorithms *random* and *round robin*.

- Taking into account the results obtained, we can affirm that there is not any significant difference in the performance between using the algorithms *two random choices* or *two random choices-round robin*.

- In the case of two-level platforms, the dispatcher of requests must not use algorithms that determine the cluster of destination of a task based on the state of the queues. It is important that the algorithm utilised distributes rapidly the tasks to the clusters, for example, by using the algorithm *round robin* and that there (at the clusters) is where the assignment is planned.

- If we wish to increase the performance of a one-level system, it is simply enough to increase the number of hosts, provided the rate of arrival of tasks, their amount and characteristics do not vary in respect to the system which is being updated. Another way to increase the performance is to transform the system into a two-level one.

- Theoretically, to make a decision, the more information on the system status the algorithm has the higher its performance should be. In reality this does not happen and for example, the algorithm *shortest queue first* is very inefficient. It is very important to study carefully the average load which the system deals with to determine whether through the use of those algorithms the performance of the system increases or decreases.

Due to what has been stated in this chapter and also in the previous ones, we can say that this project has accomplished the objectives which were set out at the beginning of the work.

I.IV.II. FUTURE WORKS

Over this section we are going to relate possible work strands which could be developed based on the present study.

Our first proposition consists of realising the same study but this time analysing the behaviour of the algorithms while making the latency of the network to vary. In this case it would be very interesting to observe how the latency influences the performance of the system and to find out the limits where increasing the bandwidth becomes irrelevant as latency becomes the key factor of the process.

A second proposition is the evaluation of the algorithms when the system has more than one dispatcher of requests available and, in addition, the algorithm of distribution of load in the dispatcher varies depending on the rate of arrival of tasks to the system.

It would also be interesting to study how the use of a core network within the cluster to which all hosts are connected influences the performance of the system, instead of

what we have done in this study where we have defined individual cables for joining every host to the router of the cluster.

We also propose to carry out diverse evaluations for different number of tasks. For all the cases in this study the evaluations have been done based always on 100.000 tasks generated; the evaluation of the created platforms and the implemented algorithms with different quantities of tasks could help to make the whole evaluation more comprehensive.

Last but not least, we propose the creation of simulations with larger platforms and with other type of algorithms.

Referencias

Andrew S. Tanenbaum, Maarten Van Steen. (2007). *Distributed Systems: Principles and Paradigms*. Disponible en: https://vowi.fsinf.at/images/b/bc/TU_Wien-Verteilte_Systeme_VO_(G%C3%B6schka)_-_Tannenbaum-distributed_systems_principles_and_paradigms_2nd_edition.pdf

Paul Krzyzanowski. (2000). *Lecture on distributed systems: A taxonomy of distributed systems*. Disponible en: https://www.cs.rutgers.edu/~pxk/rutgers/notes/content/01-intro.pdf

Henri Casanova. Recuperado en Octubre de 2015. *Simgrid: a Toolkit for the Simulation of Application Scheduling.* Disponible en: https://www.semanticscholar.org/paper/Simgrid-A-Toolkit-for-the-Simulation-of-Casanova/e928da341702c9834d5c1ab0a907c6b86627435e/pdf

Diego Chaparro González. Recuperado en octubre 2015. *Algoritmo de equilibrio de carga en un clúster heterogéneo.* Disponible en: http://viejo.dchaparro.net/doc/adcch.pdf

José Herrera Sanz. (2008). *Modelo de Programación para Infraestructuras Grid Computacionales.* Disponible en: http://eprints.sim.ucm.es/8634/1/T30914.pdf

Henri Casanova, Arnaud Legrand, Martin Quinson. (2008). *SimGrid: a Generic Framework for Large-Scale Distributed Experiments.* Disponible en: http://henricasanova.github.io/papers/casanova_uksim08.pdf

M. Baker, R. Buyya. (1999). *Cluster computing at a glance*. Disponible en: http://www.buyya.com/cluster/v1chap1.pdf

Fahad R. Dogar, Thomas Karagiannis, Hitesh Ballani, Ant Rowstron. (2015). *Decentralized Task-Aware Scheduling for Data Center Networks*. Disponible en: http://rboutaba.cs.uwaterloo.ca/Courses/CS856-F15/Presentations/decentralized_scheduling.pdf

Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, Rasit Eskicioglu, Peter Graham, Frank Sommers. Recuperado en Octubre de 2015. *Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers*. Disponible en: http://www.cloudbus.org/papers/ic_cluster.pdf

Henri Casanova. (2013). *Simulating distributed applications with SimGrid*. Disponible en: http://www.nii.ac.jp/userimg/lectures/20131010/Lecture6.pdf

Departamento ARCOS UC3M. Recuperado en octubre de 2015. *De cluster a Cloud (Visión Práctica)*. Disponible en: http://ocw.uc3m.es/ingenieria-informatica/arquitectura-de-computadores-ii/otros-recursos-1/or-f-010.-clusters-y-supercomputadores

Félix García Carballeira. Recuperado en Octubre de 2015. *Planificación y distribución de carga en sistemas distribuidos*. Disponible en: http://www.arcos.inf.uc3m.es/~dsd/lib/exe/fetch.php?media=2015-2015:planificacion.pdf

Jose Luis Bosque. (2006). *Sesión 5: Algoritmos de Equilibrio de Carga de Trabajo*. Disponible en: http://dac.escet.urjc.es/docencia/Doctorado/CPBC/sesion5.pdf

Da Simgrid Team. (2014). *SimGrid Kernel 101 Introducing the SimGrid Kernel*. Disponible en: http://simgrid.gforge.inria.fr/tutorials/simgrid-simix-101.pdf

Félix García Carballeira. Recuperado en Octubre de 2015. *Introducción a la simulación de sistemas distribuidos*. Disponible en: http://arcos.inf.uc3m.es/~dsd/lib/exe/fetch.php?media=simulacion.pdf

Da SimGrid Team. (2014). *SURF 101 Getting Started with SimGrid Models*. Disponible en: http://simgrid.gforge.inria.fr/tutorials/surf-101.pdf

Borja Bergua Guerra. (2015). *New Approaches to Data Access in Large-Scale Distributed Systems*. Disponible en: http://e-archivo.uc3m.es/handle/10016/22656#preview